MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

NESSUS, EXPLOITATION TOOLS AND PAYLOADS

INSIDE

VOL.5 NO.03
ISSUE 03/2012(32)
1898-9144

TrueNAS™ Storage Appliance: You are the Cloud

With a rock-solid FreeBSD® base, Zettabyte File System support, and a powerful Web GUI, TrueNAS™ pairs easy-to-manage software with world-class hardware for an unbeatable storage solution.

Expansion Shelves Available

TrueNAS™ 2U System

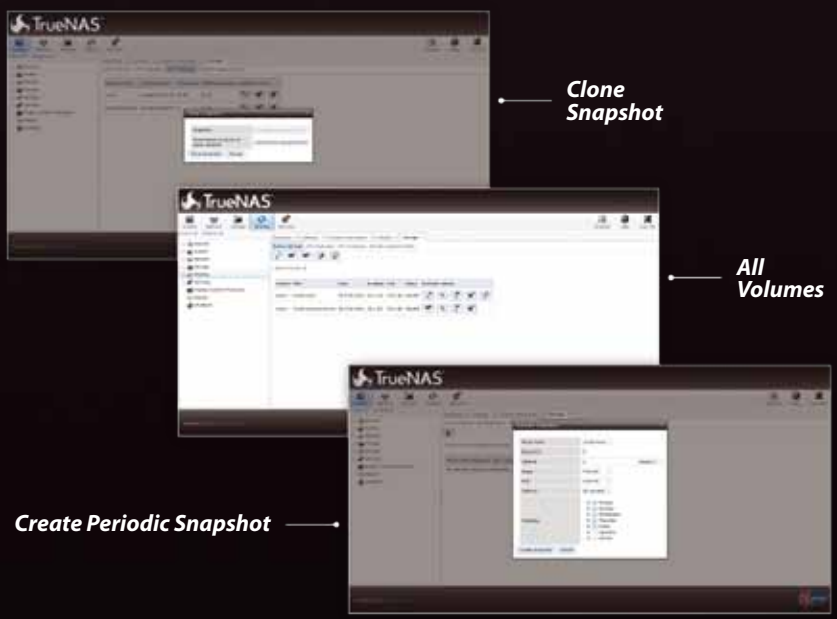
TrueNAS™ 4U System

FreeNAS 8 POWERED

Storage. Speed. Stability.

In order to achieve maximum performance, the TrueNAS™ 2U and 4U Systems, equipped with the Intel® Xeon® Processor 5600 Series, support Fusion-io's Flash Memory Cards and 10GbE Network Cards. Titan TrueNAS™ 2U and 4U Appliances are an excellent storage solution for video streaming, file hosting, virtualization, and more. Paired with optional JBOD expansion units, the TrueNAS™ Systems offer excellent capacity at an affordable price.

For more information on the **TrueNAS™ 2U** and **TrueNAS™ 4U**, or to request a quote, visit: **http://www.iXsystems.com/TrueNAS**.



TrueNAS™ 2U KEY FEATURES

- Supports One or Two Quad-Core or Six-Core, Intel® Xeon® Processor 5600 Series
- 12 Hot-Swap Drive Bays - Up to 36TB of Data Storage Capacity*
- Periodic Snapshots Feature Allows You to Restore Data from a Previously Generated Snapshot
- Remote Replication Allows You to Copy a Snapshot to an Offsite Server, for Maximum Data Security
- Software RAID-Z with up to Triple Parity
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

TrueNAS™ 4U KEY FEATURES

- Supports One or Two Quad-Core or Six-Core, Intel® Xeon® Processor 5600 Series
- 24 or 36 Hot-Swap Drive Bays - Up to 108TB of Data Storage Capacity*
- Periodic Snapshots Feature Allows You to Restore Data from a Previously Generated Snapshot
- Remote Replication Allows You to Copy a Snapshot to an Offsite Server, for Maximum Data Security
- Software RAID-Z with up to Triple Parity
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

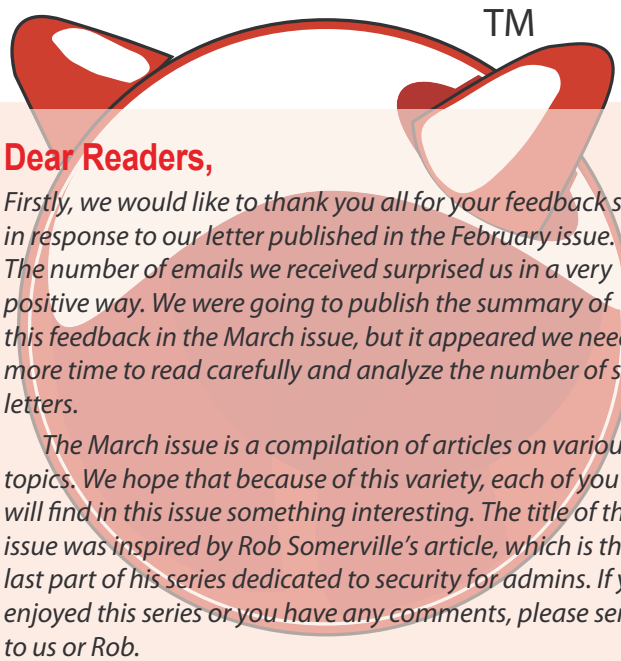
JBOD expansion is available on the 2U and 4U Systems

** 2.5" drive options available; please consult with your Account Manager*

Call iXsystems toll free or visit our website today!

1-855-GREP-4-IX | www.iXsystems.com

Powerful. Intelligent.



Dear Readers,

Firstly, we would like to thank you all for your feedback sent in response to our letter published in the February issue. The number of emails we received surprised us in a very positive way. We were going to publish the summary of this feedback in the March issue, but it appeared we need more time to read carefully and analyze the number of sent letters.

The March issue is a compilation of articles on various topics. We hope that because of this variety, each of you will find in this issue something interesting. The title of the issue was inspired by Rob Somerville's article, which is the last part of his series dedicated to security for admins. If you enjoyed this series or you have any comments, please send it to us or Rob.

In What's New Juraj Sipos described us the newest release of his project MaheshaBSD. If you are not familiar with MaheshaBSD yet, I recommend you to download it free from author's website and have some fun.

In Developers Corner this time you won't see any well known name of ours regular contributors, but you will find there a brief overview of GhostBSD. Again, if you haven't try it yet – maybe you will do it after reading this short article.

In BSD Certification series Dru Lavigne will discuss how to prepare for the BSDA certification exam. I hope it will be a helpful piece of knowledge for those who are considering taking this exam.

The rest of issue is filled with articles presenting practical knowledge. How To section will give you the opportunity to try out the described techniques and solutions. From Carlos Neira article you will find out what to do when you need to debug the program and you don't have the source code for it. Toby Richards will take you into the journey with HPC cluster called Beowulf. Luca Ferrari will show you how you can store your data with PostgreSQL.

From Giovanni Bechis' article in Tips & Tricks section you will find out how to configure OpenBSD and NPPPD to provide PPTP and L2TP VPN's in a few easy steps. This piece collected a very good reviews, so you can't miss it!

We wish you enjoy the reading and have some fun with your BSD after it!

Patrycja Przybyłowicz
& BSD Team

MAGAZINE BSD

Editor in Chief:
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Contributing:
Dru Lavigne, Toby Richards, Rob Somerville, Luca Ferrari, Nahuel Sanches, Giovanni Bechis, Jaraj Sipos, Carlos Antonio Neira

Top Betatesters & Proofreaders:
Paul McMath, Zander Hill, Bjørn Michelsen, Barry Grumbine, Imad Soltani, Eric De La Cruz, Luca Ferrari, Shayne Cardwell, Michael Dexter

Special Thanks:
Denise Ebery

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

Senior Consultant/Publisher:
Paweł Marciniak *pawel@software.com.pl*

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Executive Ad Consultant:
Ewa Dudzic
ewa.dudzic@software.com.pl

Advertising Sales:
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Publisher :
Software Press Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

Software Press Sp z o.o. SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: *editors@bsdmag.org*

What's New

Developers Corner

BSD Certification

How To

Tips & Tricks

Security

MaheshaBSD-2.0

– What's New On The Lake Manasarovar?

To readers who have not yet come across the 2010 May issue of the BSD Mag, where MaheshaBSD-1.0 was first introduced, I reiterate that MaheshaBSD is a free homemade project – a Live CD based on FreeBSD that puts together the Hindu feel and FreeBSD.

What you will learn…

- MaheshaBSD is a modular FreeBSD rescue (Live CD) toolkit (based on FreeBSD 9.0-RELEASE) and it is here introduced.

What you should know…

- Some knowledge of basic commands in FreeBSD and what to do in case of a system crash.

A few things give it this touch – for example, a possibility to use 4 keyboard layouts also with Devanagari (an Indian script used for writing Sanskrit and contemporary Indian languages) and IAST (transliteration of Sanskrit), the author's Xmodmap solution. Its name is derived from Mahesha, one of the names of Lord Shiva. The name Mahesha (MaheshaBSD) was chosen because Lord Shiva is armed with the same weapon as FreeBSD – the trident.

The Hindu feel is chosen for FreeBSD advocacy purposes (psychological tool) – that is, simply because many people who are interested in the Indian literature/history/religion will find this Live CD interesting and will learn that, in addition to Linux and Windows, they have other alternatives.

Brief Introduction Of The Project

To quickly recap what MaheshaBSD is and how it works and what it offers, the following points will shine light on what MaheshaBSD will do for you:

After you burn the ISO onto a CD, MaheshaBSD first boots into its basic MFS (Memory File System), which is independent of the CD/USB medium you booted off with. You may then eject the CD (or USB memory stick). You will be in a very rudimentary FreeBSD 9.0 system running completely in memory, which is useful for basic system tasks (fsck, copying files, mounting partitions, etc.). In this basic MFS environment you have an option to use a light version of Midnight Commander, mpg123 for playing mp3 files, but you may also run scripts and open CD/USB (file `/cdrom/usr.uzip`, which is in uzip compression). After running the opencd script, for example, the usr.uzip file gets uncompressed on the fly and mounted to `/usr`. Upon doing this, the user may start his/her X Window session (simply by typing startx; IceWM will start with the vesa video driver; however, it is important to say that the user must log in again from another console).

MaheshaBSD is a modular (and rescue) toolkit – that is, it serves like a multi purpose place with several doors

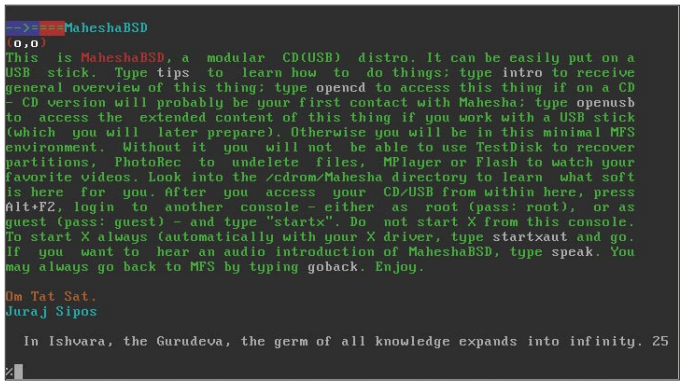


Figure 1. *When you first boot MaheshaBSD off the CD, the above brief introduction will welcome you*

and rooms you may go into and leave anytime. To clarify the concept of these doors – 1) you first boot into the MFS-only system; 2) you then mount the `usr.uzip` file on the CD with the open* commands; 3) you may go back anytime with the goback command; 4) you may put another CD/memory stick into your computer and open a different usr.uzip file on your CD/memory stick. For example, after running the opencd script, the user has an option to go back to basic MFS-only environment (that is – everything will be umounted including the usr.uzip file) and may start another *open* session by choosing from a number of available open* scripts – one of them (openclamcd) expands the CD with a very big /var directory in memory for Clamav Antivirus to work – this is important for its freshclam component, which will download these definitions from the Internet.

After downloading the virus definitions the user may scan his/her computer for viruses with the clamscan command (clamscan -r `/dir`) and then go back with the goback script to MaheshaBSD's basic MFS environment and open another uzip file.

MaheshaBSD's purpose is to bring some useful system/recovery utilities to people, but on the BSD platform – like TestDisk (which will recover lost partitions), PhotoRec (which will undelete files; it can also undelete files on USB memory sticks), Clamav (antivirus software), immediate NTFS R/W access (with ntfs-3g), chntpw (for resetting the Windows XP/W2K passwords, a very practical utility), FTP server (which immediately works without need to configure anything), MPlayer (to watch films; DivX and many other codecs are supported), and many other things – for example, MaheshaBSD can be used for presentations (you can bring it anywhere with you and show thousands of pictures to people, or present videos while giving a lecture, or watch videos with friends), or easily let your documents speak their contents for you with the MaheshaBSD's built-in speak (espeak) functionality.

Simulating The System Crash

- Your notebook falls down on the floor and the screen gets broken. You are not a techie and you do not know how to get your hard disk out. With the built-in MaheshaBSD's FTP server (vsftpd) you may log in to your computer via SSH and get to your files.
- You may run the Clamav antivirus software from within the MaheshaBSD's environment.
- You may recover lost files/partitions (TestDisk, PhotoRec).
- And many other possibilities…

What's New in MaheshaBSD-2.0?

MaheshaBSD-2.0 is based on FreeBSD 9.0-RELEASE, i386, and it was released on February 7, 2012.

MaheshaBSD-2.0 is now Skype ready – that is, you do not need anything special to install to use Skype (some Linux libraries were missing in MaheshaBSD-1.0). You just download static version of Skype from the Internet and unpack it (download it into your `/home` directory and then unpack it to /tmp because of memory limitations). *Download Static Skype* icon is placed on the IceWM's desktop.

Youtube videos now run without need to install Adobe Flash Plugin from the Internet (however, this installation is easy and the MaheshaBSD's README gives instructions how to install it). Installation of Adobe Flash Plugin is recommended only in case you want to use native version of Adobe Flash and watch youtube videos in a better quality.

X Window may now be started with the startxaut (start X automatically) script, which will generate the `/etc/X11/xorg.conf` file (with the command Xorg -configure) and the X Window GUI environment will start automatically without any manual configuration. The problem with the first (after you install FreeBSD and when you generate the `/etc/X11/xorg.conf` file with Xorg -configure) configuration of X in FreeBSD is that users must manually write the following line into `/etc/X11/xorg.conf` (into the ServerLayout Section) needed for mouse to work:

- Option „AllowEmptyImput" „off"
- The above script (startxaut) will do this work for you.

Some packages were removed, as MaheshaBSD-1.0 contained more software for the same purpose (for

Figure 2. *Youtube and Skype in MaheshaBSD-2.0*

example, mp3blaster, as cmp3 offers the same functionality). MaheshaBSD-2.0 has a new logo (Manasa Devi). MaheshaBSD-2.0 now contains a few important Hindu books with icons made for them on the IceWM's desktop (Markandeya Purana, Rig Veda, Devi Bhagavatam, and Bhagavadgita).

MaheshaBSD-2.0 has a special Xmodmap map with Devanagari and IAST support; it is in the More Progs IceWM's menu. You may use 4 keyboard layouts with it (to switch between them, use CAPSLOCK).

Seamonkey has now bookmarks for youtube videos, some Sanskrit/Hindu resources, *FreeBSD.org*, and *FreeBSD.nfo.sk*.

When you click on the Seamonkey icon, your homepage will be Startpage Privacy (*https://eu3.startpage.com/*) – a very secure search engine with Ixquick Proxy, an excellent privacy seal. Startpage is the European service that has been registered with the Dutch Data Protection Authority. Thus, users can access the Internet anonymously without need to use TOR, which is quite slow.

When you click on the xterm icon on the IceWM's desktop, you will now have a larger xterm window with larger fonts.

MaheshaBSD-2.0 saves more memory, as `/var` and `/etc` directories are now kept in the MaheshaBSD's basic MFS (/) and the opencd script does not assign any extra memory to these directories as in MaheshaBSD-1.0.

Kernel is now compressed (`/boot/kernel/kernel.gz`). MaheshaBSD-2.0 has a rewritten documentation.

A sample `wpa_supplicant.conf` file (to start wifi) is in the `/etc` directory.

MaheshaBSD-2.0 has now several more useful scripts in its `/root/bin` directory – for example, `dos2unix` (to convert TXT files in DOS format to Unix format), scpme (will copy files to and from within the MaheshaBSD's environment but via SSH), burn (an example script how to burn a CD), findmp3 (will find all mp3 files in `/mnt`, will make a play list of them and will play them with mpg123), findogg (will do the same but with ogg files), html2txt (will convert HTML files to TXT format), swapme (will make swap in memory), etc.

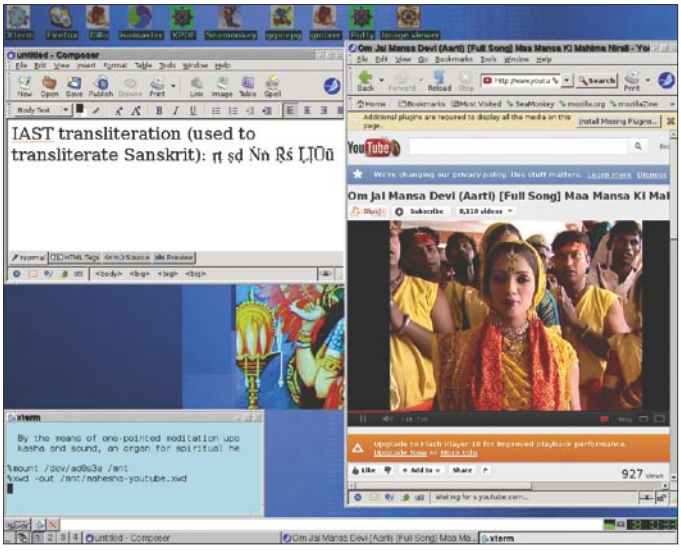


Figure 3. *Transliteration of Sanskrit with IAST*

Brief Summary Of Most Typical Features

Linux emulation is activated. You may run Skype or any Linux software under condition that you also have the necessary libraries. For that reason, the static version of Skype is recommended.

The wired Internet should work upon startup (no wifi, which you must configure manually later).

MaheshaBSD speaks. This is a very useful thing for hearing-impaired people, as running the command like `espeak -f file.txt` will give you a possibility to hear any file in TXT or HTML format (to hear HTML files, put the `-m` switch immediately after the espeak command). I made scripts that will read the documentation (tips, *README.html*, and introduction). Just type speakintro (to listen to the quick introduction of MaheshaBSD), speakreadme (to listen to the *README.html* file that contains everything important about MaheshaBSD), or speaktips (to listen to some tips).

The MaheshaBSD's modularity feature, too, is very useful – you may place a tweaked mfsroot.gz file into the MaheshaBSD's `/boot` directory (gunzip `mfsroot.gz; mdconfig -a -f mfsroot md0, mount it with mount /dev/md0 /mnt`, tweak it and gzip it back). You may then boot off your computer with MaheshaBSD and taste its several flavors: 1) router, 2) FTP server, 3) web server, etc.

The README file (it has an icon on the IceWM's desktop) instructs users how to make a USB memory stick with MaheshaBSD.

MaheshaBSD is not for everyday use. It is a recovery toolkit that can be also used for presentations, etc., and it serves this purpose only for a couple of hours. Its FTP server (vsftpd) is your door to log into any computer running MaheshaBSD (a broken notebook, for example) and save (copy) your data. You may also delete defective software on your Windows NTFS partition (to mount it in the NTFS r/w mode, use `ntfs-3g` – it works immediately).

MaheshaBSD will help you be anonymous on the Internet (with tor and polipo [a proxy server]; just click on the icon of Dillo on the IceWM's workplace and go).

You may choose national keyboard layouts in the IceWM's menu (German, Russian, Czech, Slovak); dead keys work too.

You may log into MaheshaBSD via SSH; however, only to your guest account. If you want to su to root account, you must add your guest account to the wheel group in your `/etc/group` file to allow guest to su to root, or run the script `/root/bin/sume` that will do this work for you.

You may write documents in the Seamonkey's Composer component (HTML editor). Click on the *Write documents* icon in IceWM. You can also download dictionaries and spell check your texts.

The `/boot` directory, after running the open* scripts, is mounted via `mount_nullfs` and thus all kernel modules are available.

Swap may be created with swapme scripts located in the `/root/bin` directory. Either type:

```
freecolor
```

or

```
dmesg | grep memory
```

to see how much free RAM you have, then run the following scripts:

```
swapme (to create a 100 MB swap)
swapme2 (to create a 200 MB swap)
swapme3 (to create a 300 MB swap), etc.
```

If one of them does not satisfy you, type: unswap and retry a different swapme script. What the swapme script does is:

```
mdconfig -a -t swap -s 100m -u 8
swapon -a /dev/md8
```

The above command will assign 100 MB to memory device `[/dev/md8]` and swapon will activate it as swap. You may always detach this swap with the swapoff command, for example: `swapoff /dev/md8`.

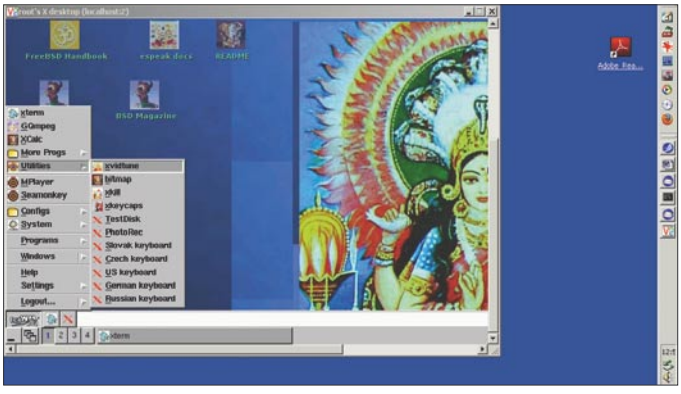


Figure 4. *MaheshaBSD running a VNC session can be also viewed on a Windows desktop*

The MaheshaBSD's Doors

The open* scripts were prepared by me and are in the `/sbin` directory in MFS. In addition to MaheshaBSD's basic MFS, the open* scripts will assign extra memory to `/root` and `/home/guest` directories. For this purpose, MaheshaBSD contains the `/mfs` directory where all important directories are kept in tgz archives: `/mfs/etc.tgz`, `/mfs/etclocal.tgz`, `/mfs/home.tgz`, `/mfs/root.tgz`, `/mfs/var.tgz`, and `/mfs/varsimple.tgz`. `/mfs/var tgz` contains the `/var/db/pkg` (packages) database and `/mfs/varsimple.tgz` has its pkg database empty.

The scripts (to open the MaheshaBSD's doors) in /sbin are:

- opencd – will mount this Live CD you booted off with (`/dev/cd0` to `/cdrom`) and the usr.uzip file on it (will be mounted to `/usr`).
- opencd2 – will do the same but with the second CD-ROM device (`/dev/cd1`).
- openclamcd – same as above, but the script will assign extra memory to the `/var` directory; this is needed to make room for the Clamav virus definitions that must be downloaded from the Internet (into `/var/db/clamav`); the `/var` dir is made in memory with more than 100 MB for that purpose.
- openclamcd2 – will do the same, but with the second CD-ROM device.
- openclamusb – will open the USB memory stick (`/dev/da0s1a`) but with no usr.uzip mounted to `/usr`; you must have a fully populated `/usr` directory on your USB memory stick, which is particularly good for installing/deinstalling packages; the `/var` directory can carry all Clamav virus definitions if downloaded from the Internet.
- openclamusb2 – will do the same thing but with the second USB device (`/dev/da1s1a`).
- openclamusbuzip – same as above, but with `usr.uzip` mounted to `/usr`.
- openclamusbuzip2 – same as above but with the second USB device (`/dev/da1s1a`).
- openda0 – a script for preparation of a USB memory stick in the MaheshaBSD's environment (after you run it, you then just need to copy all MaheshaBSD's files from `/cdrom` onto your USB memory stick and you will thus have a fully working MaheshaBSD on a memory stick – read the README file on the MaheshaBSD's IceWM desktop for additional information).

- openda1 – same as above but with the second USB device (`/dev/da1s1a`).
- opendvd – same as opencd, but the script mounts usrdvd.uzip (it is expected that you make it yourself later; read the MaheshaBSD's *README.html*) instead of usr.uzip; you will thus have, after going back to the MaheshaBSD's basic MFS with the goback script, a possibility to mount a much bigger uzip file than usr.uzip on the CD.
- opendvd2 – same as above but with the second CD-ROM device (`/dev/cd1`).
- openmincd – this script will mount the `usr.uzip` file on the MaheshaBSD's CD with minimal memory assigned to `/dev/md` devices (the script assigns only 10 MB to the `/tmp` directory), which is good for systems with low hardware resources.
- openmincd2 – same as above but with the second CD-ROM device.
- openusb – this will open your memory stick (`/dev/da0s1a`) with fully populated `/usr` dir on your stick (`usr.uzip` is not mounted to `/usr`).
- openusb2 – will do the same but with the second USB device.
- openusbuzip (or ouz) – will mount your memory stick (`/dev/da0s1a` to `/usb`) and usr.uzip is mounted to `/usr`.
- openusbuzip2 (or ouz2) – will do the same but with the second USB device.
- goback – will umount everything and the user returns to basic MaheshaBSD's MFS as in the situation he/she booted off with this Live CD (or USB memory stick) the first time and did not run any open* script.

Memory Requirements

To see memory disks attached to the system as configured devices in FreeBSD, type (in the console): `mdconfig -l`.

MaheshaBSD first goes into its basic MFS environment (in the `root` directory `/`). It is about 50 MB in size (mounted as `/dev/md0` to `/`) – a very simple (stripped) system without the fruits of the standard FreeBSD `/usr` contents. In this MFS – that is, before you run the opencd script (and other open* scripts), you work only with 54 MB completely in memory with a few free megabytes left (5.8 MB), which is not enough to download Skype and other goodies (like Adobe Flash Plugin). All directories in it are writable.

After running the opencd script, the following directories will be made in memory (other scripts may bring different results):

```
/tmp 60 MB
/root 50 MB
/usr/local/etc 35 MB
/usr/home 45 MB
/usr/local/lib/npapi/linux-f10-flashplugin 14 MB
```

RAM totally 204 MB + 54 MB (basic MFS) = 258 MB

However, if you do not have the above memory available, you can always run the openmincd script, which creates only 10 MB for the `/tmp` directory – that is, 64 MB of RAM should suffice.

All open* scripts (except for openda0 and openda1) mount `/cdrom/usr.uzip` (or `/usb/usr.uzip`) to `/usr` (`/usr/local/etc`, `/usr/home/guest` and `/usr/local/lib/npapi/linux-f10-flashplugin` are made writable in memory). When mounted, the `/usr` dir has the size of 1.5 GB (uncompressed), although the file usr.uzip (compressed) has only 583 MB.

Conclusion

MaheshaBSD is free software but copyrighted. The copyright only pertains to the work made by me and not to packages, as licenses of these have their own conditions. The idea behind the MaheshaBSD project is to support and spread words about FreeBSD. Its Hindu touch serves the same purpose, because there are still many people who have never heard of FreeBSD. If they search for some Hindu keywords, they may possibly find it and try it and convince their neighbors that FreeBSD is not only for the techies.

In the future, MaheshaBSD will always keep its original contours, because a possibility to type wise ideas in Sanskrit or IAST transliteration will make many people look out of their Window(s) where today, unfortunately, also Linux belongs.

I thank *www.rootbsd.net* for allowing me to distribute MaheshaBSD.

JURAJ SIPOS

Juraj lives in Slovakia and he works in a library in an educational institute. Some time in the past he was fortunate to travel around the world and he spent a bit of time in India and Australia. Juraj's hobbies are computers, mostly Unix, but spirituality too. His first published computer article was Xmodmap Howto (http://tldp.org/HOWTO/Intkeyb/). In addition to computers, he is very interested in Hinduism but not really the guru side of things, but more-so freedom and self-actualization. More at his website: http://www.freebsd.nfo.sk/ (FreeBSD) http://www.freebsd.nfo.sk/maheshaeng.htm (MaheshaBSD)

Great Specials

On FreeBSD & PC-BSD Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.0 Jewel Case CD Set
or FreeBSD 9.0 DVD

$29.95
PC-BSD 9.0 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.0 DVD

$99.95
The FreeBSD CD **or** DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.0 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Hoodies
Stay Warm in Pullovers & Zip Ups

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.0 Jewel Case CD/DVD $39.95

CD Set Contains:

- **Disc 1:** Installation Boot LiveCD (i386)
- **Disc 2:** Essential Packages Xorg, GNOME2 (i386)
- **Disc 3:** Installation Boot LiveCD (amd64)
- **Disc 4:** Essential Packages Xorg, GNOME2 (amd64)

FreeBSD 8.2 CD $39.95
FreeBSD 8.2 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.0 ..... $29.95
FreeBSD Subscription, start with DVD 9.0 ..... $29.95
FreeBSD Subscription, start with CD 8.2 ..... $29.95
FreeBSD Subscription, start with DVD 8.2 ..... $29.95

PC-BSD 9.0 DVD (Isotope Edition)

PC-BSD 9.0 DVD $29.95
PC-BSD Subscription $19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.0 ..... $79.95

PC-BSD 9.0 Users Handbook $24.95

BSD Magazine $11.99

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD & PCBSD Caps $20.00

BSD Daemon Horns $2.00

Bundle Specials!
Save $$$

Just Plain Fun
Mousepads & Novelty Horns

BSD Magazine
Available Monthly

For even **MORE** items
visit our website today!

www.FreeBSDMall.com

GhostBSD: A Brief Overview

My name is Nahuel Sanchez, co-founder of GhostBSD. I will gladly give you all the information you need to know about the GhostBSD project.

GhostBSD was created to encourage the use of FreeBSD users with little experience, and also for those curious they need / want to learn freebsd in a simple, or for those seeking a more robust alternative to the current options available in Linux kernels (either for safety for stability or for licenses). An operating system with graphical environment, simple and useful, as is implemented in GhostBSD, it helps enthusiasts to take their first steps, provides more security and incentive to experiment, at first but then the graphical interface with options for system configuration finished adapting the code to their needs.

The goals of the GhostBSD projects is to:

- encourage the use of BSD in client's terminals (in commercials) so as to augment awareness on the use of Open Source software alternatives (both for flexibility and for cost reductions)
- provide an excellent and respectable alternative to the field of open operating systems
- promote the use of Open Source software (such as web browsers, word processors, email clients and so on)
- spread the use of BSD on desktop computers.

Who Are Involved in GhostBSD?

GhostBSD was born in the FreeBSD forums. At present, Eric lives in Dieppe NB Canada and I live in Rosario, Argentina. But, although we live so far apart from each other, we are in regular contact working on the project and connecting ourselves by means of emails, IM, and the project´s IRC channel (as well as newsletter for our followers).

TAll the same, the project has been enriched by important partners have collaborated with interesting and qualified modifications. Nevertheless, we have gone on collecting opinions, suggestions, and all types of consults through our web site (*http://ghostbsd.org*), for which we are deeply thankful to you all and be sure that we will take account of each of them so as to enhance the project.

Messages via emails as well as comments left in our site, replies and post in our forums or conversations via the IRC

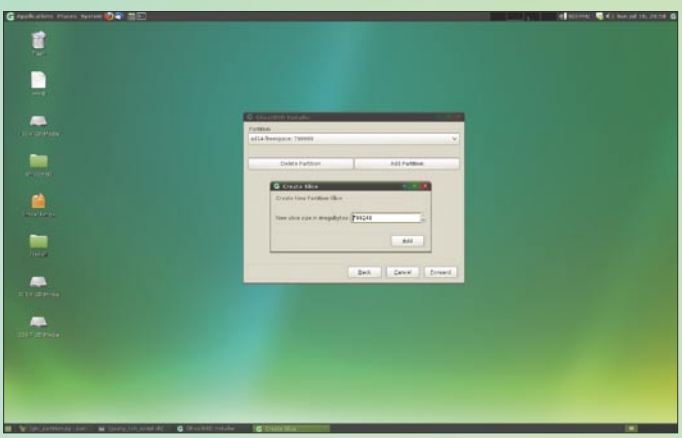


Figure 1. *Installer*

Figure 2. *Version-1.5*

Figure 3. *Version-2.0*

channel (FreeNode##GhostBSD) are closely studied and assumed as valuable material for the evolution of the project.

How GhostBSD Is Economically Solvent?

Since it's inceptions, GhostBSD is kept in an internal cannel of distribution. Torrents and *SourceForge.net* were mainly used until we had the opportunity to rent a server (VPS) for direct download.

The project is still alive, thanks to three input sources:

- One it's the capital can be met in order to defray the costs through their own contribution, as well as through donations at cants via PayPal (by entering in our page) and also by anybody who wish to advertise in our site.
- Money donations that allow us to pay hosting and other cost.
- If you have some knowledge about programming and want to help us with our task, please contact us. We are always in a need of enthusiastic people who want to share their ideas and participate in the project.

History of GhostBSD

Version 1.0 was released in March 2010. It was based on FreeBSD 8 and used GNOME 2.28.

Version 1.5 based on FreeBSD 8.1, uses GNOME 2.30. Compiz. The German bimonthly released magazine freeX (1/2011) featured GhostBSD 1.5 on a supplemented DVD and in an article.

Figure 4. *Version-2.3*

Version 2.0 is based on FreeBSD 8.2, and was released on March 13, 2011. Some changes in version 2 include improvements to GDM and bug fixes.

Version 2.5 of the final release of GhostBSD is based on the official FreeBSD 9.0 and is out since Jan 24, 2012. This version of GhostBSD has two main branches of the system – one is based on the GNOME desktop, the other on the LXDE desktop. Both are available in amd64 and i386 versions and in form of installable CD/DVD or USB images. Since that month, Jan 2012, a detailed wiki-guide *How to build GhostBSD*? in combination with the GhostBSD toolkit is published, to build a personal customized version of the GhostBSD installation image, adding all the packages not found in the official FreeBSD releases, actual FreeBSD 9.0 (per january 2012). The GhostBSD toolkit has been designed to allow building of both, i386 and amd64 architectures on amd64 based computer systems with at least 4GB of disk space to swap, a sincere computing power and FreeBSD installed on.

If want a comparison tablet you can found one here: *http://en.wikipedia.org/wiki/Comparison_of_BSD_operating_systems*.

Short-term targets

One of the primary objectives with GhostBSD, is to implement a software to install packages without ports (the implementation of a Network Manager).

In other words, this provides and update software and new release. The project goals is to have a standard of the Gnome with FreeBSD and being friendly to the new user.

NAHUEL SANCHEZ
Co-founder, web, External Affairs
http://ghostbsd.org/

ERIC TURGEON
Founder and developer
http://ghostbsd.org/

How Do I Study for the BSDA Certification?

The previous article in this series addressed some common misconceptions about certification and described why you should be BSDA certified. This article will discuss how to prepare for the BSDA certification exam.

The previous article in this series discussed the concern: *there aren't any training materials available or the training materials are too expensive*. It explained that a psychometrically valid examination assesses real world skills and why the exam's objectives are the ultimate study resource.

This article describes how to prepare for the BSDA examination in practical terms. It describes the steps one can take to obtain those "real world skills" and to determine when one is ready to take the exam.

When studying for the BSDA, the following steps are recommended:

Download the BSDA Certification Requirements Document

Since the audience definition, domain percentages, and exam objectives are the *roadmap* used to create an examination, the document containing that information is your study roadmap. Finding and downloading this document should be your first step when studying for any certification exam. The document containing this information for the BSDA is entitled the *BSDA Certification Requirements Document* and is available for download in the following languages:

- English: *http://www.bsdcertification.org/downloads/pr_20051005_certreq_bsda_en_en.pdf*
- Spanish: *http://www.bsdcertification.org/downloads/pr_20051005_certreq_bsda_es_mx.pdf*
- Russian: *http://www.bsdcertification.org/downloads/pr_20051005_certreq_bsda_ru_ru.pdf*

This document is divided into three sections:

Section 1 Contains

- *the definition of audience for BSDA:* this is a detailed description describing the level of experience required to pass the examination. When studying for the exam, remember that questions can not be harder than those that can be answered by the intended audience.
- *the operating system versions covered by the BSDA:* this section indicates that the candidate needs a basic knowledge of 4 BSD operating systems. When setting up your study lab, you can install any version from the lowest number listed up to and including the most recent release version. For example, when installing FreeBSD, you can install any version from 4.11 (the lowest listed version) up to 9.0 (the highest RELEASE version as of this writing).
- *re-certification requirements:* in order to meet accreditation requirements, a certification can not be for life. In other words, it must have an expiry date. BSDA certifications are valid for a period of 5 years. The BSDA re-certification requirements will be published by Q3, 2012.

Section 2

Contains a description of the 7 study domains. The percentages for the study domains are listed at *http://www.bsdcertification.org/certification/associate.html*. Table 1 lists the study domains, their percentage, and the number of exam objectives within each domain. Note that the number of objectives may not match the weighting as weighting indicates the importance of that

domain within the exam while the number of objectives indicates the number of testable tasks within that domain.

Section 3
Contains the objectives themselves, divided by domain. The next section will demonstrate how to read the exam objectives and use them for study purposes.

Appendix A
Contains an alphabetized list of all of the commands and files listed in the exam objectives. It also maps each command/file to the 4 BSD operating systems as some commands/files are not available in every BSD.

Read the Exam Objectives

The exam objectives (Section 3) begin with a section entitled *Using the BSDA Study Domains*. Read this section carefully as it contains detailed advice on how to use the exam objectives.

Each objective has four components:

- *number*: where the second number indicates the domain and the third number the objective within that domain. For example, 3.1.2 is the second objective in domain 1 (Installing & Upgrading the OS and Software). Domain 1 has 10 objectives and is worth 13% of the exam.
- *objective*: a detailed task to be assessed. As indicated in *Using the BSDA Study Domains*, watch for verbs (which require you to know how to do something) v.s. *recognize* (which requires you to know the name of a file or command).
- *concept*: a detailed description of what the candidate is expected to know about that objective.
- *practical*: the commands or files associated with the objective. These are also listed alphabetically in Appendix A.

As an example, here are two exam objectives:

3.2.8 Recognize BSD firewalls and rulesets.
Concept
Each BSD comes with at least one built-in firewall. The BSDA candidate should recognize which firewalls are available on each BSD and which commands are used to view each firewall's ruleset.

Practical
ipfw(8), ipf(8), ipfstat(8), pf(4), pfctl(8) and firewall(7)

3.4.1 Create, modify and remove user accounts.
Concept
Managing user accounts is an important aspect of system administration. The BSDA should be aware that the account management utilities differ across BSD systems and should be comfortable using each utility according to a set of requirements.

Practical:
vipw(8); pw(8), adduser(8), adduser.conf(5), useradd(8), userdel(8), rmuser(8), userinfo(8), usermod(8), and user(8)

The first example is the eighth objective in Domain 2 (Securing the Operating System). It begins with *recognize*, meaning that the user is not expected to know how to configure a BSD firewall or ruleset, but instead needs to be able to recognize the available tools. The concept clearly indicates what you need to recognize: which firewalls are available and which commands are used to view a firewall's ruleset. The practical clearly indicates the names of the man pages representing the applicable firewalls and commands. When studying this objective, review any man pages that you are unfamiliar with and compare the commands listed in the practical to Appendix A so that you can recognize which commands apply to which BSD. Since this objective only requires you to know how to view, don't memorize or get mired in the details of the (fairly lengthy) man pages as you read through them. Instead, focus on how to view a ruleset for each firewall.

Table 1. *BSDA Study Domains*

Domain	Weighting	Number of Objectives
1. Installing & Upgrading the OS and Software	13%	10
2. Securing the Operating System	11%	13
3. Files, Filesystems, and Disks	15%	14
4. Users and Accounts Management	16%	9
5. Basic System Administration	12%	24
6. Network Administration	15%	15
7. Basic Unix Skills	17%	17

The second example is the first objective in Domain 4 (Users and Account Management). It uses the verbs *create, modify,* and *remove user accounts*, indicating that the user needs to demonstrate experience in how to perform those three actions. The concept clearly indicates that the utilities vary by BSD and the practical lists the possible tools. This means that you should use Appendix A to determine which tools match which BSD, then practice each tool in your lab setup until you are comfortable using each tool to create, modify, and remove user accounts.

Make a List

As you read through the objectives, start to organize them in order to determine which skills need to be learned and how much study will be required. You may want to print out the document in order to write notes next to each objective. Alternately, you may find it easier to start a list that organizes the objectives into roughly three categories:

- know it
- wouldn't hurt to review this
- need to learn how to do this

You should find that most of the objectives that start with *recognize* and which you don't already know, can go into the second category. Objectives that start with a verb and which vary by BSD will probably fit into the third category. You may wish to further separate the *recognize* objectives (which require some reading) from the *verb* objectives (which require some practice) in order to get a better idea of how much lab practice time will be involved.

Once the objectives are categorized, you have your personalized study action plan. You will know exactly which man pages you should review and which commands you need to learn how to use. You can then decide how many objectives you can tackle at a time and calculate a rough estimate on how long it will take you to become comfortable with the material covered by the exam objectives.

It is recommended that you print out Appendix A and mark the commands that you need to review or learn how to use. Once you have worked your way through those commands, you are more than ready to take and pass the BSDA certification exam!

Setup Your Study Lab

When studying, you will be reading man pages, comparing their contents to what is required by a specific exam objective, and practicing commands. You do not need access to a BSD system in order to read man pages as each BSD provides online man pages:

- FreeBSD: *http://www.freebsd.org/cgi/man.cgi*
- NetBSD: *http://netbsd.gw.com/cgi-bin/man-cgi*
- OpenBSD: *http://www.openbsd.org/cgi-bin/man.cgi*
- DragonFly BSD: *http://leaf.dragonflybsd.org/cgi/web-man*

Online man pages provide a convenient way to compare the same man page for each BSD simultaneously using a tabbed web browser. The online versions also contain hyperlinks to other man pages mentioned in the SEE ALSO section, making it easy to quickly learn more about a topic that interests you.

In order to practice commands, you will need access to each BSD operating system. Each BSD can be downloaded for free from that project's website. You do not need multiple machines in your study lab, as each BSD can be installed as a guest within a virtual environment. Possible virtual environments include:

VMWare

Free, commercial product. Downloads for Windows and Linux are available from *http://www.vmware.com/products/player/*.

Virtualbox

Free, open source application. Downloads for Windows, Mac OS X, Linux, and Solaris are available from *https://www.virtualbox.org/wiki/Downloads*. BSD versions are available as ports, packages, and PBIs. Easy to use, but requires a good amount of RAM if you will be running multiple BSD guests at the same time.

qemu

Free, open source application. Command line by default, but GUI versions (aqemu, kqemu, and qemu-launcher) are also available. Allows you to run multiple BSD guests with minimal RAM requirements. BSD versions are available as ports, packages, and PBIs.

When setting up your virtual environment, you will need to configure the network interface as a bridged adpater in order to access the network using the guest operating system.

To assist you in quickly creating a study lab, the BSD Certification Group offers a BSDA Study DVD. This DVD is updated every 6 months or so to the latest RELEASE version of each operating system. The current version of the DVD contains the following:

- FreeBSD 8.2, including ports collection
- NetBSD 5.1, including pkgsrc
- OpenBSD 5.0, including packages
- DragonFly BSD 2.10.1 including pkgsrc
- BSDA Exam Objectives (pdf)
- BSDA Command Reference (pdf)
- Psychometrics Explained (pdf)
- BSDA Task Analysis Survey Report (pdf)
- BSD Usage Survey Report (pdf)
- BSDA Test Delivery Survey Report (pdf)
- BSDP Job Task Analysis Survey Report (pdf)
- BSDP Certification Requirements (pdf)
- FreeBSD Handbook (pdf)
- FreeBSD FAQs (pdf)
- The Complete FreeBSD (pdf)
- NetBSD Guide (pdf)
- DragonFly BSD Guide (pdf)
- pkgsrc Guide (pdf)
- OpenBSD FAQ (pdf)
- Latest draft of the wiki version of the BSDA Study Guide (pdf)
- Detailed instructions on how to setup the lab environment and networking using qemu/aqemu

It should be noted that each of the items on the DVD is freely available from the BSD project and BSD certification websites. The DVD is meant to be a convenience as well as a way to support BSD certification as all proceeds are used to pay for the ongoing psychometric maintenance of the exam. DVDs can be purchased for $40 USD + shipping from *http://www.bsdcertification.org/store/*.

Get Your Questions Answered

Once you have prepared your study action plan and configured your lab setup, you need to find the time to review and learn the objectives until you understand them and can accomplish the required tasks. Most of this learning can be achieved with practice, but occasionally you will come across something that you are not sure about.

Like most open source projects, the BSD Certification project is comprised of a large community of volunteers who share a common interest (in this case, system administration of BSD operating systems). Several resources are available if you have a question regarding the understanding of an exam objective:

- *IRC*: the `#bsdcert` channel is available on IRC Freenode.
- *LinkedIn*: a LinkedIn group of working professionals who are interested in BSD certification is available at *http://www.linkedin.com/groups/BSD-Certification-1600767*. Once you become BSDA certified, you can also join the LinkedIn group for BSDA certified professionals (*http://www.linkedin.com/groups?gid=1600807*).
- *Facebook*: if you use Facebook, you can join the BSD certification community at *https://www.facebook.com/groups/55432547309/*. Exam events are also listed here as they are arranged.
- *study wiki*: a wiki where volunteers contribute tips for each objective is available at *http://bsdwiki.reedmedia.net/wiki/Table_of_Contents.html*. If you would like to contribute to the wiki, you can request the registration password on the bsdcert IRC channel or Facebook group.

Even if you don't encounter any questions while studying, you are welcome to join the BSD certification community using any of these resources.

Summary

This article provided practical tips for preparing for the BSDA examination. Once you have finished reviewing and practicing the exam objectives, you are ready to take the exam.

The next article in this series will describe where to take the exam and how to arrange for an exam if there currently isn't an examination event or testing center near your location.

DRU LAVIGNE

Dru Lavigne is author of BSD Hacks, The Best of FreeBSD Basics, and The Definitive Guide to PC-BSD. As Director of Community Development for the PC-BSD Project, she leads the documentation team, assists new users, helps to find and fix bugs, and reaches out to the community to discover their needs. She is the former Managing Editor of the Open Source Business Resource, a free monthly publication covering open source and the commercialization of open source assets. She is founder and current Chair of the BSD Certification Group Inc., a non-profit organization with a mission to create the standard for certifying BSD system administrators, and serves on the Board of the FreeBSD Foundation.

GDB(1) and Truss for Debugging

Sometimes you are lucky to have the source code for the program you need to debug. However, there are times when the source code isn't available.

What you will learn…

- A technique for debugging programs without source code
- How to see system calls invoked by a process
- Some basic gdb(1) commands

What you should know…

- Some assembly language for x86

When all hell is breaking loose, what do you do? On your unix machine there are tools that can save the day. OpenBSD, FreeBSD and NetBSD all have the ktrace utility for following the various kernel related activities of a given process. FreeBSD has a tool specifically for tracing system calls. It's called truss(1) and when used together with gdb(1) it can give you a clearer view into a black box.

This is not specifically a truss(1) tutorial; you can check the man page for truss(1) for more details; here we are just scratching the surface (Figure 1).

Let me give you an idea of what truss(1) can do. As the man page says, truss(1) traces all the system calls invoked by the specified process we want to look at. Let's see – I have moused daemon in my unix box, let's check it out.

First we need to obtain the PID for the moused daemon and then just type:

```
TRUSS(1)              FreeBSD General Commands Manual              TRUSS(1)

NAME
     truss -- trace system calls

SYNOPSIS
     truss [-faedDS] [-o file] [-s strsize] -p pid
     truss [-faedDS] [-o file] [-s strsize] command [args]

DESCRIPTION
     The truss utility traces the system calls called by the specified process
     or program.  Output is to the specified output file, or standard error by
     default.  It does this by stopping and restarting the process being moni-
     tored via ptrace(2).

     The options are as follows:

     -f      Trace descendants of the original traced process created by
             fork(2), vfork(2), etc.
```

Figure 1. *Truss manpage extract*

```
truss -p <pid of the process you want to take a look>
```

We are looking right now at the syscalls and their arguments (Figure 2).

You want to know the return value of the syscall? or check if something is wrong? You can use gdb(1) for that! You don't have the source code? No problem, you can look at the registers. The return value of most system calls and program functions is stored in the `%eax` register (I am referring to x86 architecture).

I have written a small program that we will use as an example. It simply outputs the sum of the variables in a `for()` loop – pretty simple but enough for this proof of concept. Here is the code: Listing 1.

Save this code to a file. I called it test.c (very original). If you have installed make(1) and a C compiler, you just need to type:

```
read(3,"\0",1)                                  = 1 (0x1)
select(1024,{3},0x0,0x0,0x0)                    = 1 (0x1)
read(3,"\^A",1)                                 = 1 (0x1)
select(1024,{3},0x0,0x0,0x0)                    = 1 (0x1)
read(3,"\^A",1)                                 = 1 (0x1)
select(1024,{3},0x0,0x0,0x0)                    = 1 (0x1)
read(3,"\0",1)                                  = 1 (0x1)
select(1024,{3},0x0,0x0,0x0)                    = 1 (0x1)
read(3,"\0",1)                                  = 1 (0x1)
select(1024,{3},0x0,0x0,0x0)                    = 1 (0x1)
read(3,"\^?",1)                                 = 1 (0x1)
clock_gettime(12,{4039.831020110 })             = 0 (0x0)
ioctl(4,CONS_MOUSECTL,0xbfbfeb8c)               = 0 (0x0)
^C[root@VirtualHost ~]# truss -p 616
SIGNAL 17 (SIGSTOP)
gettimeofday({1327111096.062437 },0x0)          = 0 (0x0)
sendto(6,"<28>Jan 21 02:58:16 moused: fail"...,70,0x0,NULL,0x0) = 70 (0x4e)
```

Figure 2. *Truss moused output*

```
# make test
```

or as usual, do a

```
# cc test.c -o test
```

(notice that I have omitted the -g flag so we don't have any debug information generated).

Now that you have compiled the source let's inspect this with gdb(1).

```
# gdb test
```

We set our first break point at the main function. As you recall, we don't have the source code for this, so we are blindfolded and looking for any clue that might help us resolve a problem.

```
(gdb) b main
```

We type r (run) and start the program flow...

Listing1. *Test.c example source code*

```
#include<stdio.h>


    int dummy(p1,p2,p3)
    {
         int tmp;
             tmp=p1+p2+p3;
         printf("dummy value: %d\n",tmp);
         return tmp ;

    }

int main()
{
    int p1=1,p2=2,p3=3,i,tmp;

    for(i=0;i<=10;i++)
    {
        tmp=dummy(p1,p2,p3);
            printf("dummy() returned %d\n",tmp);
        p1++;
        p2++;
        p3++;
    }
}
```

Listing 2. *Using gdb disas command for dumping of assembler code for function main*

```
(gdb) disas

Dump of assembler code for function main:
0x080483d0 <main+0>:    lea    0x4(%esp),%ecx
0x080483d4 <main+4>:    and    $0xfffffff0,%esp
0x080483d7 <main+7>:    pushl  -0x4(%ecx)
0x080483da <main+10>:   push   %ebp
0x080483db <main+11>:   mov    %esp,%ebp
0x080483dd <main+13>:   push   %ecx
0x080483de <main+14>:   sub    $0x34,%esp
0x080483e1 <main+17>:   movl   $0x1,-0x18(%ebp)
0x080483e8 <main+24>:   movl   $0x2,-0x14(%ebp)
0x080483ef <main+31>:   movl   $0x3,-0x10(%ebp)
0x080483f6 <main+38>:   movl   $0x0,-0xc(%ebp)
0x080483fd <main+45>:   jmp    0x804843e <main+110>
0x080483ff <main+47>:   mov    -0x10(%ebp),%eax
0x08048402 <main+50>:   mov    %eax,0x8(%esp)
0x08048406 <main+54>:   mov    -0x14(%ebp),%eax
0x08048409 <main+57>:   mov    %eax,0x4(%esp)
0x0804840d <main+61>:   mov    -0x18(%ebp),%eax
0x08048410 <main+64>:   mov    %eax,(%esp)
0x08048413 <main+67>:   call   0x80483a4 <dummy>

0x0804841b <main+75>:   mov    -0x8(%ebp),%eax
0x0804841e <main+78>:   mov    %eax,0x4(%esp)
0x08048422 <main+82>:   movl   $0x8048510,(%esp)

0x08048429 <main+89>:   call   0x80482d8 <printf@plt>

0x0804842e <main+94>:   addl   $0x1,-0x18(%ebp)
0x08048432 <main+98>:   addl   $0x1,-0x14(%ebp)
0x08048436 <main+102>:  addl   $0x1,-0x10(%ebp)
0x0804843a <main+106>:  addl   $0x1,-0xc(%ebp)
0x0804843e <main+110>:  cmpl   $0xa,-0xc(%ebp)
0x08048442 <main+114>:  jle    0x80483ff <main+47>
0x08048444 <main+116>:  add    $0x34,%esp
0x08048447 <main+119>:  pop    %ecx
0x08048448 <main+120>:  pop    %ebp
0x08048449 <main+121>:  lea    -0x4(%ecx),%esp
0x0804844c <main+124>:  ret
End of assembler dump.
```

```
(gdb) r

Breakpoint 1, 0x080483de in main ()
Current language: auto; currently asm
#0  0x080483de in main ()
```

So we hit our break point. We can only can see the asm instructions in this program, so we type: Listing 2.

The hexadecimal values in the column on the far left are the addresses of instructions which will be executed as the program runs. These values will probably differ from what you'll see if you're running this code. We can use these addresses for setting breakpoints.

In the output we see a function called `dummy()`. Let's put a break point there (Listing 3).

So here we see that this function calls the classic `printf()` function and puts the return code in the `$eax` register.

To see the contents of all registers just type: Listing 4.

We step through the `dummy()` function until we pass the call to `printf()`. We can then inspect the return value by typing:

```
(gdb) p $eax
```

To modify the integer value passed to the `printf()` function we set a break point at the instruction that pushes the `%eax` register onto the stack and then change the value in the register. To do this, we need to use the hexadecimal address when setting the breakpoint. This is the instruction where we want execution to stop:

```
0x080483bb <dummy+23>:  mov   %eax,0x4(%esp)
```

so we set the break point using the hexadecimal address of the instruction:

```
(gdb) break *0x080483bb
```

Now we can modify the `%eax` register value to 99 and let the program continue running.

```
(gdb) set $eax=99
(gdb) c

dummy value: 99
(additional gdb output ignored)
dummy() returned 6
```

Listing 3. *Setting a breakpoint at the dummy function*

```
(gdb) b dummy
Breakpoint 2 at 0x80483aa

Let's check the asm for the dummy function.
(gdb) disas dummy
Dump of assembler code for function dummy:
0x080483a4 <dummy+0>:   push   %ebp
0x080483a5 <dummy+1>:   mov   %esp,%ebp
0x080483a7 <dummy+3>:   sub   $0x18,%esp
0x080483aa <dummy+6>:   mov   0xc(%ebp),%edx
0x080483ad <dummy+9>:   mov   0x8(%ebp),%eax
0x080483b0 <dummy+12>:  add   %edx,%eax
0x080483b2 <dummy+14>:  add   0x10(%ebp),%eax
0x080483b5 <dummy+17>:  mov   %eax,-0x4(%ebp)
0x080483b8 <dummy+20>:  mov   -0x4(%ebp),%eax
0x080483bb <dummy+23>:  mov   %eax,0x4(%esp)
0x080483bf <dummy+27>:  movl   $0x8048510,(%esp)
0x080483c6 <dummy+34>:  call   0x80482d8 <printf@plt>
0x080483cb <dummy+39>:  mov   -0x4(%ebp),%eax
0x080483ce <dummy+42>:  leave
0x080483cf <dummy+43>:  ret
```

Listing 4. *Dumping the registers content*

```
(gdb) i r

 Breakpoint 2, 0x080483aa in dummy ()
eax            0x2    2
ecx            0x0    0
edx            0xb7f8f0f0    -1208422160
ebx            0xb7f8dff4    -1208426508
esp            0xbf9bb410    0xbf9bb410

esi            0x8048460    134513760
edi            0x80482f0    134513392
eip            0x80483aa    0x80483aa <dummy+6>
eflags         0x282    [ SF IF ]
cs             0x73    115
ss             0x7b    123
ds             0x7b    123
es             0x7b    123
fs             0x0    0
gs             0x33    51
```

The first line of output shows that `printf()` used the value in the `%eax` register (99). The value of the variable tmp did not change, as can be seen by the output when `printf()` is called from the `main()` function (`tmp = 6` in this particular iteration).

To modify the return value of the `dummy()` function in the program, we need to change the value of the `%eax` register just before exiting the function. This is done by setting a breakpoint at the *leave* instruction.

```
0x080483cb <dummy+39>:  mov   -0x4(%ebp),%eax
0x080483ce <dummy+42>:  leave  <-- here !!
0x080483cf <dummy+43>:  ret
```

As before, we need to set the breakpoint using the address of the instruction.

```
(gdb) break *0x080483ce
(gdb) set $eax=999
```

Now, when the program continues,the return value of `dummy()` is output by `printf()` as:

```
dummy() returned 999
```

You can modify the flow of the program using the registers as the program checks for a file; or in the case of a routine that returns an error code, you can override the result easily.

So it is basically wash, rinse and repeat until you find the bug or the condition that triggers the failure and is giving you a headache. This is just the tip of the iceberg, but it is pretty helpful playing with the registers and truss if you don't own the application that is causing the problems.

I hope this was helpful. It's brief but the documentation is always the best source for information. Just calling up the ol' man page can show you new possibilities to explore apart from these.

CARLOS ANTONIO NEIRA

Carlos Antonio Neira is a C, Unix and Mainframe developer. He develops in asm and does some kernel development for a living. In his free time he contributes to open source projects. Apart form that, he spends his time on testing and experimenting with his machines. What gives him a great fun is solving the old problems with new ideas.

PostgreSQL:

MVCC and Vacuum

In the previous article readers have seen how to quickly install and configure a PostgreSQL cluster, as well as how to do logical backups, using pg_dump(1) and physical backup (with particular regard to Point In Time Recovery).

What you will learn…

- what MVCC is and how it is exploited in PostgreSQL
- how to deal with Vacuum and Auto-Vacuum

What you should know…

- basic SQL concepts
- how to configure and access a PostgreSQL instance
- basic shell commands

This article shows a little more about PostgreSQL internals and how it exploits MVCC for high concurrency. Readers will also learn about the importance and usage of *vacuum* for regular maintanance. The database used in the examples can be rebuilt at any time using the simple script in Box 1.

MVCC

MVCC stands for Multi-Version Concurrency Control and is a technique that PostgreSQL uses to provide high concurrency while keeping database consistency. Giving a full explanation of how MVCC works is out the scope of this article, please see the official manual and the references for further readings.

Box 1. *Content of the magazine.sql text file used to reload the data (file magazine.sql).*

```
BEGIN;
CREATE TABLE IF NOT EXISTS magazine(pk serial NOT NULL,
id text,
month int,
issuedon date,
title text,
PRIMARY KEY(pk),
UNIQUE (id)
);
TRUNCATE TABLE magazine;
INSERT INTO magazine (pk, id, month, issuedon, title)
VALUES(1,'2012-01', 1,       '2012-01-01'::date ,     'FreeBSD: Get Up To Date');
INSERT INTO magazine (pk, id, month, issuedon, title)
VALUES(2,'2011-12', 12,       '2012-01-01'::date , 'Rolling Your Own Kernel');
INSERT INTO magazine (pk, id, month, issuedon, title)
VALUES(3,'2011-11', 11,       '2012-01-01'::date, 'Speed Daemons');
COMMIT;
```

Databases must ensure that even when clients are executing concurrent statements on the same set of data, the latter remains consistent. This usually requires locking: briefly the first client that gets access to the data locks it so that other clients have to wait until the lock is released to get access to the same data. Locking can quickly become a bottleneck and can lower the concurrency of the queries. MVCC takes another approach to the problem: each client has access to a *private snapshot* of the data. Trhough snapshots multiple versions of the same data (e.g., the same tuples) are available to concurrent clients. This does not eliminate locks at all, but dramatically reduces the need for them in the backend. Simplifying MVCC can be compared to *Copy-On-Write* (COW) filesystems like ZFS: each time a tuple is going to be manipulated a clone is created and changes are applied to the latter.

PostgreSQL numbers each transaction with a 32 bit progressive identifier called xid; moreover within each transaction each statement is also progressively identified (*cid*). It is worth reminding that each statement is always executed in a transaction context, either explic (i.e., issuing a `BEGIN`) or implic (no `BEGIN` has been issued).

PostgreSQL keeps track of MVCC attaching to every tuple metadata fields: *xmin*, *xmax*, *cmin* and *cmax*. Such fields are available to the user but are hidden in each `SELECT` statement until not explicitly named (see Listing 1). The meaning of each metadata field is the following:

- *xmin* indicates the transaction identifier that created the tuple;
- *xmax* indicates the transaction identifier that invalidated or is going to invalidate the tuple (via `UPDATE` or `DELETE`);
- *cmin* indicates the command identifier within a transaction that created the tuple;
- *cmax* indicates the command identifier within a transaction that invalidated the tuple (i.e., either updated or deleted the tuple), if the tuple has been invalidated.

To get an idea of how metadata is stored, consider the query shown in Listing 1: you can see that all the tuples have been created by the same transaction 727, with three consecutive `INSERT`s (*cmin* and *cmax* range from 0 – first command – to 2 – last command within the transaction) and has not been yet invalidated (i.e., *xmax* is still 0). You can also see that the transaction 727 was executed 5 transactions before the current one: the `age()` function returns the distance (in terms of transactions) from the current transaction to another transaction identifier and the function `txid_current()` returns the identifier for the current transaction (therefore the `SELECT` statement is executed as implicit transaction 732). In other words, five transactions ago there was an explicit transaction numbered 727 that loaded the three shown tuples (with three consecutive statements) and nothing more changed such tuples. Now consider doing an implicit transaction that updates the tuple with *pk = 1*, as reported in the second half of Listing 1. What happens is that the tuple with *xmin* 727 was substituted by the new copy of the same tuple

Listing 1. *Evaluating MVCC data for each tuple.*

```
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), * FROM magazine;
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), * FROM magazine LIMIT 3;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  727 |    0 |    0 |    0 |   5 |          732 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  727 |    1 |    0 |    1 |   5 |          732 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  727 |    2 |    0 |    2 |   5 |          732 |  3 | 2011-11 |    11 | Speed Daemons

bsdmagdb=# UPDATE magazine SET title = 'FreeBSD: Get Up To Date!' WHERE pk = 1;
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM magazine;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  727 |    1 |    0 |    1 |   7 |          734 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  727 |    2 |    0 |    2 |   7 |          734 |  3 | 2011-11 |    11 | Speed Daemons
  733 |    0 |    0 |    0 |   1 |          734 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date!
```

(with data opportunely changed by the UPDATE) and now the tuple has *xmin* 733 (one transaction before the last SELECT in Listing 1) and both *cmin* and *cmax* are set to 0 (the first command in an implicit transaction is the statement itself). What happened behind the scenes? As shown in Figure 1 the old tuple (*xmin* 727) was marked as no more valid (*xmax* 733) and a new tuple has been created (*xmin* 733). When you execute a SELECT on a table PostgreSQL gives you back only tuples marked as still valid. To better understand what *valid* means we have to do a little more experimentation. To that purpose it is worth installing the *pageinspect* extension (avilable from the *-contrib* module) which allows the DBA to see how a data page is actually used; it is also worth installing the *pgstattuple* module to get even more information about the data page status. Listing 2 shows how to install the above modules in your own 9.1 database. As readers can see from Listing 2, the data page contains four tuples while the table in Listing 1 shows only three of them: the trick is that the first tuple of Listing 2 is the one inserted by transaction 727 and made not valid from transaction 733 (*lp = 1*), which substituted it with another tuple (*lp = 4*). As soon as the system realizes that tuples are only three (or the DBA informs the database to adjust the data page – more on this later) the system cleans the tuple in the data page to free space (see Listing 3).

Table 1 shows the transaction isolation levels as defined by the SQL Standard and how PostgreSQL adhere to that; please note that it is fine for the standard to handle a isolation level with a stricter one and therefore before version 9.1 PostgreSQL provided support only for two levels, *READ COMMITTED* and *SERIALIZABLE*, making the other two, *READ UNCOMMITTED* and *REPEATABLE READ* respectively an alias of the formers. Since 9.1 a new level has been natively supported, *REPEATABLE READ*, that behaves exactly as the *SERIALIZABLE* level in previous versions. As readers can see snapshots are computed at different times depending on the transaction isolation level: for

Listing 2. *Installing the pageinspect and pgstattuple modules.*

```
~> find /usr/local/lib -name 'pageinspect.so'
/usr/local/lib/postgresql/pageinspect.so
~> find /usr/local/lib -name 'pgstattuple.so'
/usr/local/lib/postgresql/pgstattuple.so
bsdmagdb=# CREATE EXTENSION pageinspect;
bsdmagdb=# CREATE EXTENSION pgstattuple;
bsdmagdb=# SELECT lp, lp_flags, t_xmin::text::int8 AS xmin, t_xmax::text::int8 AS xmax, t_ctid
FROM heap_page_items( get_raw_page('magazine', 0) )
ORDER BY lp;
 lp | lp_flags | xmin | xmax | t_ctid
----+----------+------+------+--------
  1 |        1 |  727 |  733 | (0,4)
  2 |        1 |  727 |    0 | (0,2)
  3 |        1 |  727 |    0 | (0,3)
  4 |        1 |  733 |    0 | (0,4)
```

Listing 3. *Inspecting the data page after the database removed expired tuples.*

```
bsdmagdb=# SELECT lp, lp_flags, t_xmin::text::int8 AS xmin, t_xmax::text::int8 AS xmax, t_ctid
FROM heap_page_items( get_raw_page('magazine', 0) )
ORDER BY lp;
 lp | lp_flags | xmin | xmax | t_ctid
----+----------+------+------+--------
  1 |        1 |  727 |    0 | (0,1)
  2 |        1 |  727 |    0 | (0,2)
  3 |        1 |  733 |    0 | (0,3)
```

Table 1. *Transaction isolation levels in PostgreSQL.*

Transaction Isolation Level	Transaction Isolation Problems (and their brief description)			Natively Supported by PostgreSQL	Snapshot refers to
	Dirty Read	Non-Repeatable Read	Phantom Read		
	An unfinished transaction can read data manipulated by another concurrent transaction even if the latter has not yet committed.	An unfinished transaction read data that is then manipulated by another committed transaction, so that the former can no more read the same data again.	An unfinished transaction re-executes a query that returns a different set of tuples due to another concurrent transaction that committed changes that affected the selection criteria.		
Read Uncommited	Possible	Possible	Possible	NO, default to READ COMMITTED	
Command to set isolation level		SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;			
Read Committed	Not possible	Possible	Possible	YES, via READ COMMITTED	Command start.
Command to set isolation level		SET TRANSACTION ISOLATION LEVEL READ COMMITTED;			
Repeatable Read (default)	Not Possible	Not Possible	Possible	NO (before 9.1), default to SERIALIZABLE YES (since 9.1), via REPEATABLE READ	Transaction start.
Command to set isolation level		SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;			
Serializable	Not Possible	Not Possible	Not Possible	YES, via SERIALIZABLE	Transaction start.
Command to set isolation level		SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;			

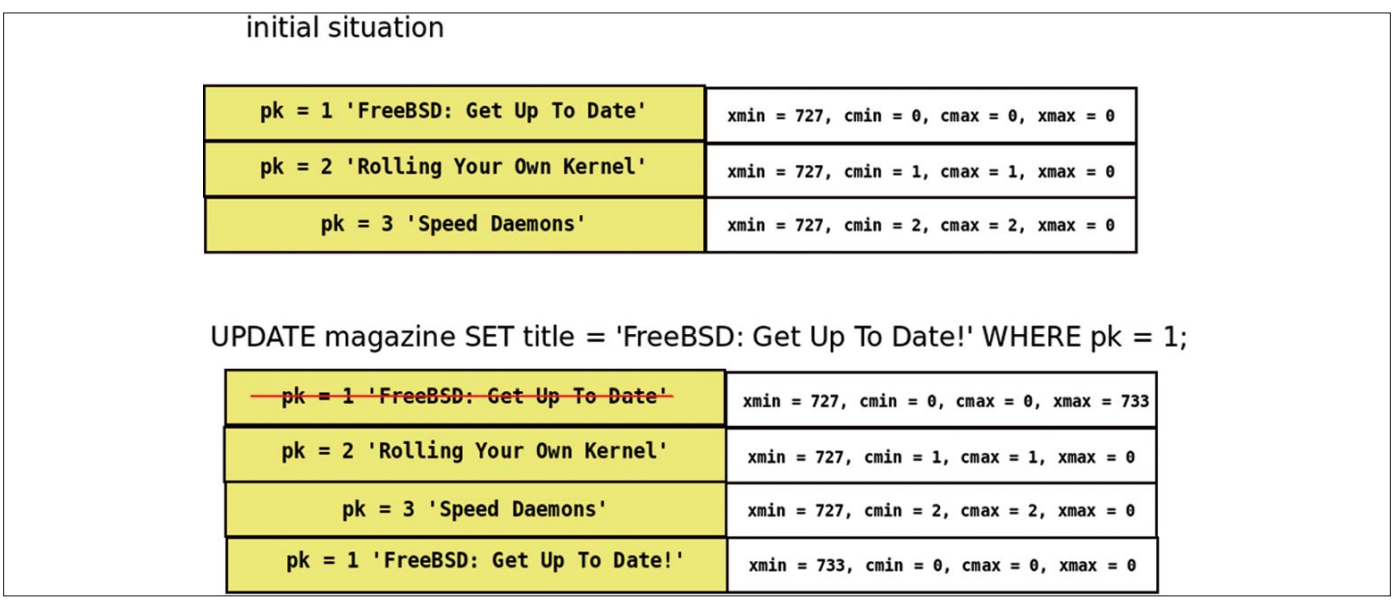


Figure 1. *Conceptual visualization of the changes performed in Listing 1*

Listing 4. *MVCC and concurrent transactions (READ COMMITTED)*

(in terminal A)

```
bsdmagdb=# BEGIN;
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM magazine;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  754 |    0 |    0 |    0 |   3 |          757 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  755 |    0 |    0 |    0 |   2 |          757 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  756 |    0 |    0 |    0 |   1 |          757 |  3 | 2011-11 |    11 | Speed Daemons
(3 rows)

bsdmagdb=# UPDATE magazine SET title = '[SOLD OUT] ' || title WHERE id like '2011-%';
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM magazine;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |               title
------+------+------+------+-----+--------------+----+---------+-----------------------------------
  754 |    0 |    0 |    0 |   3 |          757 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  757 |    0 |    0 |    0 |   0 |          757 |  3 | 2011-11 |    11 | [SOLD OUT] Speed Daemons
  757 |    0 |    0 |    0 |   0 |          757 |  2 | 2011-12 |    12 | [SOLD OUT] Rolling Your Own Kernel


bsdmagdb=# SELECT lp, lp_flags, t_xmin::text::int8 AS xmin, t_xmax::text::int8 AS xmax, t_ctid
FROM heap_page_items( get_raw_page('magazine', 0) )
ORDER BY lp;
 lp | lp_flags | xmin | xmax | t_ctid
----+----------+------+------+--------
  1 |        1 |  754 |    0 | (0,1)
  2 |        1 |  755 |  757 | (0,5)
  3 |        1 |  756 |  757 | (0,4)
  4 |        1 |  757 |    0 | (0,4)
  5 |        1 |  757 |    0 | (0,5)
bsdmagdb=# COMMIT; - after this B is unlocked!
```

(in terminal B)

```
bsdmagdb=# BEGIN;

bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM magazine;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  754 |    0 |    0 |    0 |   4 |          758 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  755 |    0 |  757 |    0 |   3 |          758 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  756 |    0 |  757 |    0 |   2 |          758 |  3 | 2011-11 |    11 | Speed Daemons
(3 rows)

bsdmagdb=# UPDATE magazine SET title = title || ' [SOLD OUT]' WHERE id like  '2011-%';
- the transaction is locked here until A does a COMMIT/ROLLBACK!
```

a *READ COMMITTED* a new snapshot is computed each time a command begins in order to ensure that the command will see all the data committed by other transactions; on the other hand in *REPEATABLE READ* and *SERIALIZABLE* mode a snapshot is created once when the transaction is started, so that it will see data committed only before the transaction itself. The difference between *REPEATABLE READ* and *SERIALIZABLE* in version 9.1 is that the former uses a lock on the data to avoid concurrent manipulations, therefore simulating a sequential execution of the transactions, while the latter keeps a set of so called *predicate locks*, which are locks on queries and not on their data. Predicate locks are used by PostgreSQL to understand if transactions are executing conflicting queries, forcing then one to abort without having to lock the data (and therefore providing a better concurrency). In order to help the system incrementing the concurrency it is also possible to indicate a transaction as read-only or write-only via the `SET TRANSACTION ISOLATION` command.

In order to see how MVCC works with concurrent transactions clean and refill the *magazine* table, then start two transactions in two different terminals (A with *xid* 757 and B with *xid* 758); see Listing 4 for details. Imagine that A executes the `UPDATE` before the one of B; since the default isolation level is *READ COMMITTED* then B has to wait for A to either commit or rollback, therefore the B `UPDATE` keeps the session locked waiting for A to conclude. Please note, as shown in the B terminal, that B sees the old version of the data (i.e., the data has not been modified by A permanently) but it is also informed that transaction A is changing the data (*xmax* is set to 757). In other words, B knows that the tuples will expire if transaction 757 (A) commits. Inspecting the page data readers can see how there are two tuples with a *xmax* set to 757 and two new tuples with *cmin* and no *cmax* set.

Replaying the same experiment with a fresh situation and making transactions A and B serializable will report an error in transaction B because the `UPDATE` cannot be serialized.

Listing 5. *MVCC and transaction commands*

```
bsdmagdb# BEGIN;
bsdmagdb=# \i magazine.sql
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), * FROM magazine;

 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  784 |    0 |    0 |    0 |   0 |          784 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  784 |    1 |    0 |    1 |   0 |          784 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  784 |    2 |    0 |    2 |   0 |          784 |  3 | 2011-11 |    11 | Speed Daemons

bsdmagdb=# DECLARE cursor_mvcc CURSOR FOR SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM
                  magazine;
bsdmagdb=# UPDATE magazine SET title = '[SOLD OUT] ' || title WHERE id like '2011-%';
bsdmagdb=# SELECT xmin, cmin, xmax, cmax, age(xmin), txid_current(), id, title FROM magazine;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |               title
------+------+------+------+-----+--------------+----+---------+-----------------------------------
  784 |    0 |    0 |    0 |   0 |          784 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  784 |    3 |    0 |    3 |   0 |          784 |  3 | 2011-11 |    11 | [SOLD OUT] Speed Daemons
  784 |    3 |    0 |    3 |   0 |          784 |  2 | 2011-12 |    12 | [SOLD OUT] Rolling Your Own Kernel

bsdmagdb=# FETCH ALL FROM cursor_mvcc;
 xmin | cmin | xmax | cmax | age | txid_current | pk |   id    |          title
------+------+------+------+-----+--------------+----+---------+-------------------------
  784 |    0 |    0 |    0 |   0 |          784 |  1 | 2012-01 |     1 | FreeBSD: Get Up To Date
  784 |    1 |  784 |    1 |   0 |          784 |  2 | 2011-12 |    12 | Rolling Your Own Kernel
  784 |    0 |  784 |    0 |   0 |          784 |  3 | 2011-11 |    11 | Speed Daemons
```

To understand the usage of *cmin* and *cmax* we have to issue conflicting commands within the same transaction. To do so we use a `CURSOR`, that is a resource that will fetch row by row data from a set. As Listing 5 shows, a transaction is started, then the tuples are loaded and a cursor to query the table is declared. Since the cursor is the third command in the transaction, further commands will not change the snapshot seen by the cursor: therefore an `UPDATE` of the tuples is immediately reflected in the transaction, but not in the cursor. In this scenario the in-transaction snapshot is based on the values of *cmin* and *cmax*. Of course, it does not make sense to compare *cmin* and *cmax* out of a transaction boundaries, since the transaction isolation level will define how snapshots are visible. As an implementation detail, it is worth noting that *cmin* and *cmax* are internally stored as a single value, so such information does not suffice to understand if a multi-statement transaction has created and expired a tuple. The solution PostgreSQL adopts is to keep track in the memory page of a so-called *combo command id* that informs that the tuple has been created and expired within the transaction.

Anatomy of a Data Page

PostgreSQL data is contained in so called *data-pages* (see Figure 2): each page has a space where tuples are placed that grows toward low addresses and an array of pointers to each tuple, called *linear pointers* (lp), which grows towards high addresses. When a specific tuple has to be found, PostgreSQL loads (if not already present) the data page that contains such tuple into a free shared buffer (a region of shared memory) and inspects it to find the linear pointer that leads to the tuple. The advantage of keeping linear pointers within the data page is that tuples can be re-arranged within the page without having to change the way to find the page itself.

Listing 6 provides a simple shell script that simulates a workload to see how data pages change during several tuple operations. The workload is quite simple: starting from an empty *magazine* table, it inserts a set of tuples, immediately modifies them and finally deletes all of them. From a user perspective the *magazine* table is unchanged at the end of the workload, because it is empty. However the script reports the following output:

```
==========================
Filenode was /postgresql/cluster1/base/16398/17110
Size before starting: 0
Size after insert:    262144
     [ 32 pages with 223893 bytes for 5000 live tuples]
Size after update:    540672
     [ 66 pages with 258893 bytes for 5000 live tuples]
                    (around 2 times initial size)
Size after delete:    540672 [ 66 pages ] (around 2 times
                    initial size)
==========================
```

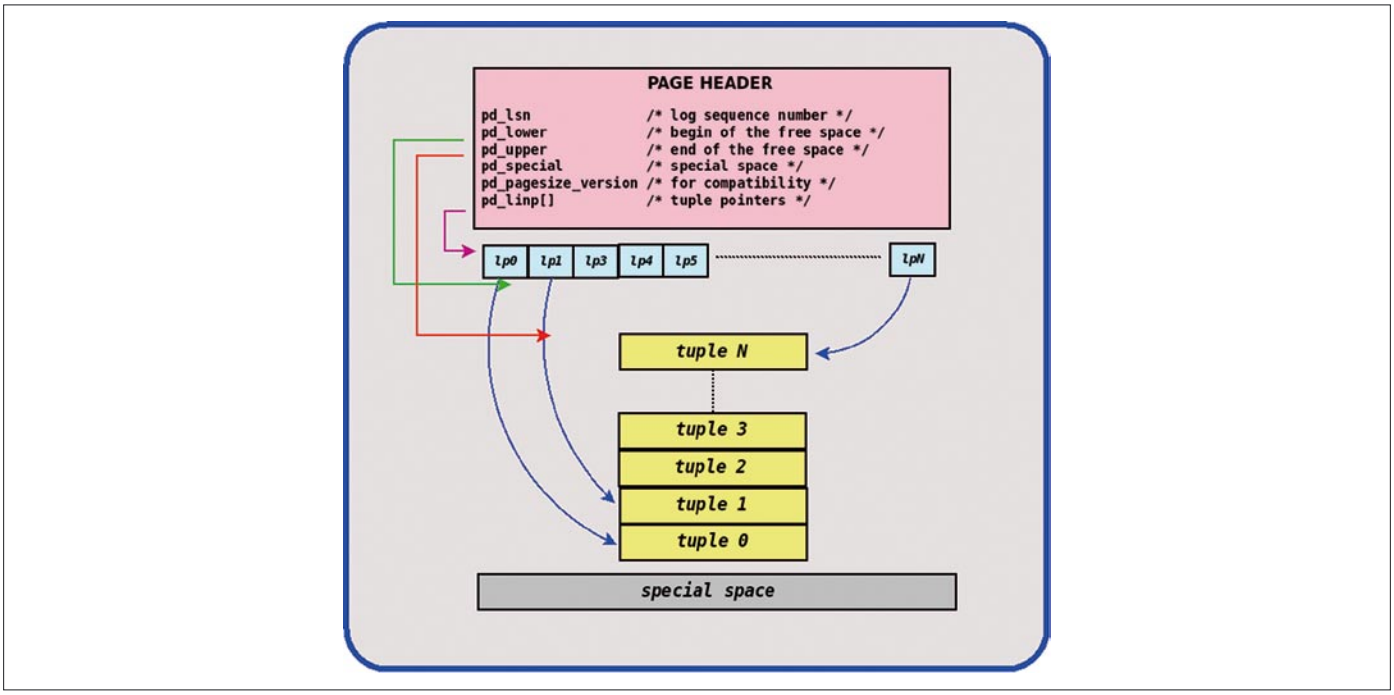


Figure 2. *PostgreSQL data page layout*

Listing 6. *A script to test MVCC and how data pages changes*

```
#!/bin/sh
PGDATA=/postgresql/cluster1
DBOID='oid2name -q | grep bsdmagdb | awk '{print $1;}''
PAGE_SIZE='expr 8 '*' 1024'
FILENODE=${PGDATA}/base/${DBOID}/${TABLEOID}
FILLFACTOR=100
PAGE_QUERY=" SELECT lp, lp_flags, t_xmin::text::int8 AS xmin, t_xmax::text::int8 AS xmax, t_ctid FROM heap_page_
                    items( get_raw_page('magazine', 0) ) ORDER BY lp;"
echo "Cleaning the magazine table..."
psql -U bsdmag -c "TRUNCATE TABLE magazine;" bsdmagdb
psql -U bsdmag -c "VACUUM FULL magazine;" bsdmagdb
psql -U bsdmag -c "ALTER TABLE magazine SET (fillfactor=$FILLFACTOR);" bsdmagdb
TABLEOID='oid2name -U bsdmag -t magazine -d bsdmagdb -q | awk '{print $1;}''
FILENODE=${PGDATA}/base/${DBOID}/${TABLEOID}
ls -lh $FILENODE
SIZE_0='ls -lk $FILENODE | awk '{print $5;}''
sleep 2
echo "Inserting tuples..."
psql -U bsdmag -c "INSERT INTO magazine(id, title) VALUES( generate_series(1, 5000), 'vacuum-test');" bsdmagdb
psql -U bsdmag --pset pager=off -c "${PAGE_QUERY}" bsdmagdb
ls -lh $FILENODE
SIZE_1='ls -lk $FILENODE | awk '{print $5;}''
SIZE_TUPLE_1='psql -U bsdmag -A -t -c "SELECT tuple_len FROM pgstattuple('magazine');"
SIZE_COUNT_1='psql -U bsdmag -A -t -c "SELECT tuple_count FROM pgstattuple('magazine');" bsdmagdb'
sleep 2
echo "Updating tuples.."
psql -U bsdmag -c "UPDATE magazine SET title = 'UPDATED' || title; " bsdmagdb
psql -U bsdmag --pset pager=off -c "${PAGE_QUERY}" bsdmagdb
ls -lh $FILENODE
SIZE_2='ls -lk $FILENODE | awk '{print $5;}''
SIZE_TUPLE_2='psql -U bsdmag -A -t -c "SELECT tuple_len FROM pgstattuple('magazine');" bsdmagdb'
SIZE_COUNT_2='psql -U bsdmag -A -t -c "SELECT tuple_count FROM pgstattuple('magazine');" bsdmagdb'
sleep 2
echo "Deleting tuples.."
psql -U bsdmag -c "DELETE FROM magazine; " bsdmagdb
psql -U bsdmag --pset pager=off -c "${PAGE_QUERY}" bsdmagdb
ls -lh $FILENODE
SIZE_3='ls -lk $FILENODE | awk '{print $5;}''
echo "=========================="
SIZE_1_T='expr $SIZE_1 / $SIZE_1'
SIZE_1_P='expr $SIZE_1 / $PAGE_SIZE'
SIZE_2_T='expr $SIZE_2 / $SIZE_1'
SIZE_2_P='expr $SIZE_2 / $PAGE_SIZE'
SIZE_3_T='expr $SIZE_3 / $SIZE_1'
SIZE_3_P='expr $SIZE_3 / $PAGE_SIZE'
echo "Filenode was $FILENODE"
echo "Size before starting: $SIZE_0"
```

From the output readers can clearly see that the table started as empty, then we added data for 32 pages and after the update the relation doubled its data pages. That is because the inserted tuples were marked as expired from the UPDATE, and so a new copy of each tuple was inserted as new. Finally, after the DELETE also the second copy of each tuple was marked as expired; this is the reason why the table storage retained its size (see Figure 3). The script of Listing 6 reports also a dump of the first and last page data as follows:

```
==========================
Dump of the first data page
 lp | lp_flags | xmin | xmax | t_ctid
----+----------+------+------+--------
  1 |        3 |      |      |
  2 |        3 |      |      |
  3 |        3 |      |      |
  4 |        3 |      |      |
  5 |        3 |      |      |

==========================
Dump of the last data page 65
 lp | lp_flags | xmin | xmax | t_ctid
----+----------+------+------+--------
  1 |        1 | 1026 | 1027 | (65,1)
  2 |        1 | 1026 | 1027 | (65,2)
  3 |        1 | 1026 | 1027 | (65,3)
  4 |        1 | 1026 | 1027 | (65,4)
  5 |        1 | 1026 | 1027 | (65,5)
==========================
```

It is interesting to note that tuples in the first data pages are now marked as *dead* (flag = 3) while tuples in the last page (i.e., the last version of the deleted tuples) are marked as *living* (flag = 1); this means that tuples in the first page have not to be considered at all by running transaction, while tuples in the last page must be considered according to the snapshot visibility rules.

Careful readers should have noted that the ouput of Listing 6 shows that data pages after an UPDATE have not exactly doubled, but are now a little more than the initial number (i.e., 66 versus 32 initial pages). The reason for this *extra* space is that the UPDATE changed the size of each tuple (changing the *title* column), as reported by the total size of *live tuples*.

Vacuum

As explained in previous sections, when a tuple is modified (either via UPDATE or DELETE) a new version of the tuple is stored. This fills data pages with *old and no*

Listing 6b. *A script to test MVCC and how data pages changes*

```
echo "Size after insert:    $SIZE_1 "
echo "     [ $SIZE_1_P pages with $SIZE_TUPLE_1 bytes for $SIZE_COUNT_1 live tuples]"
echo "Size after update:    $SIZE_2 "
echo "     [ $SIZE_2_P pages with $SIZE_TUPLE_2 bytes for $SIZE_COUNT_2 live tuples] (around $SIZE_2_T times initial
                  size)"
echo "Size after delete:    $SIZE_3 [ $SIZE_3_P pages ] (around $SIZE_3_T times initial size)"
echo "=========================="

echo "=========================="
echo "Dump of the first data page"
psql -U bsdmag --pset pager=off -c "${PAGE_QUERY}" bsdmagdb
echo "=========================="
echo "=========================="
LAST_PAGE='expr $SIZE_3_P - 1'
echo "Dump of the last data page $LAST_PAGE"
PAGE_QUERY_LAST=" SELECT lp, lp_flags, t_xmin::text::int8 AS xmin, t_xmax::text::int8  AS xmax, t_ctid FROM heap_
                  page_items( get_raw_page('magazine', $LAST_PAGE ) ) ORDER BY lp LIMIT 5;"
psql -U bsdmag --pset pager=off -c "${PAGE_QUERY_LAST}" bsdmagdb
echo "=========================="
```

more visible tuples. A special tool, called *vacuum*, can clean up no more visible tuples freeing storage space for new live data. Vacuum is a kind of swiss knife for PostgreSQL and can act over a single table or an entire database and has several aims. It is worth noting that *vacuum* can be invoked from a live connection or via the `vacuumdb(1)` (and its brother `vacuumlo(1)`) command line executable. There are several *flavours* of vacuum, mainly:

standard: reclaim for free space but within data page boundaries (so will not free effective space until the last data page can be entirely erased);

- *full*: is the most aggressive way of running it, and it will clean all the expired tuples in all the data pages, reorganizing living tuples;
- *analyze*: used to update the internal statistics (used for instance by the query optimizer). Can be run even on a single column;
- *freeze*: used to avoid xid wraparound (see later).

Please note that the `vacuum full` locks the entire table(s), and therefore is the most aggressive and less concurrent maintenance task. The script of Listing 7, if executed immediately after the one of Listing 6, provides the following output due to a vacuum full:

```
==========================
Filenode was /postgresql/cluster1/base/16398/25307
Size before starting: 540672
Size after VACUUM:    0 [ 0 pages ] (around 0 times
                    initial size)
==========================
```

As readers can see the table is now empty and the data storage is empty too. While running, the `vacuum` command reports:

```
INFO:  vacuuming „public.magazine”
INFO:  „magazine”: found 5000 removable, 0 nonremovable
                    row versions in 66 pages
```

which states that the expired 5000 tuples are going to be removed from the storage since they are no more visible to any running transaction.

There is another important task that `vacuum` has to accomplish: avoid the *xid wraparound*. As explained in the previous sections, each transaction is identified by an unique progressive number, the *xid*, which is internally handled as a 32 bit integer. Sooner or later, the *xid* will wrap around and since tuples are visible to transactions with a lower *xid* than the running one, the database will be in a condition where younger transactions will have a *xid* that is lower (so in the past) that of older running transactions. To avoid this, PostgreSQL starts numbering transactions from a non-zero value (3) and `vacuum` *freezes* old committed transaction tuples with a *xid* equal to *frozen-xid* (2), so that such tuples will always be perceived in the past even after a xid wrap around. The effects of the *freeze* can be seen executing a `vacuum` manually on the *magazine* table:

```
bsdmagdb=# VACUUM FREEZE magazine;
bsdmagdb=# SELECT xmin, cmin, cmax, xmax FROM magazine LIMIT 5;
 xmin | cmin | cmax | xmax
------+------+------+------
    2 |    0 |    0 |    0
```

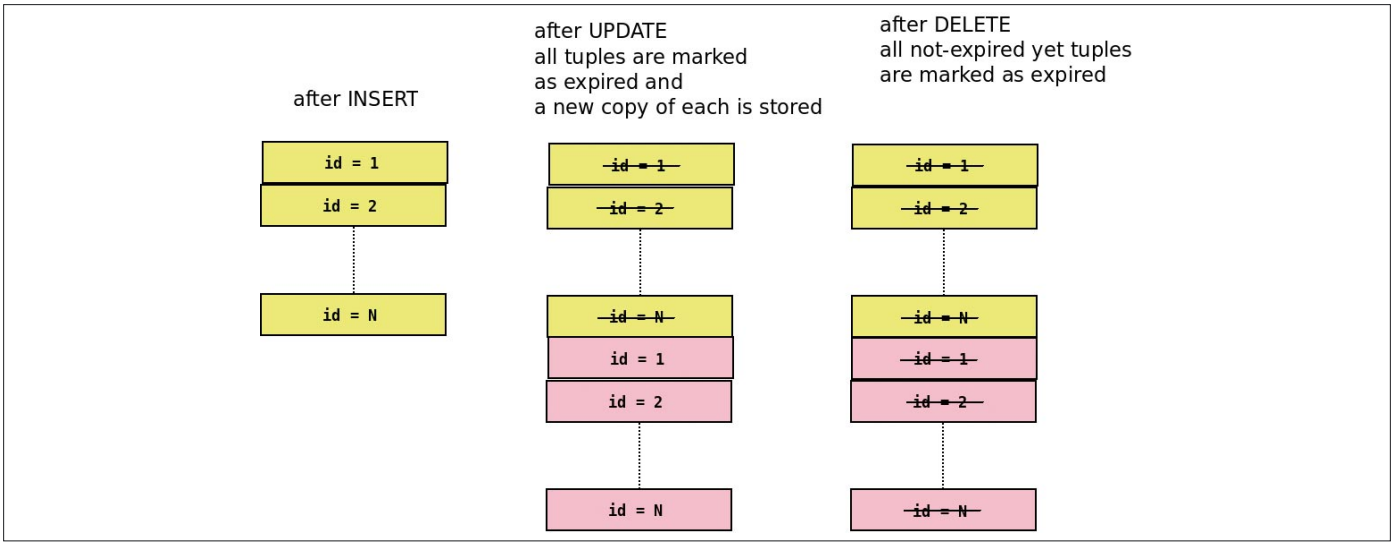


Figure 3. *Conceptual evolution of tuples within the magazine table while executing the example workload*

```
 2 |     0 |     0 |     0
 2 |     0 |     0 |     0
 2 |     0 |     0 |     0
 2 |     0 |     0 |     0
```

To avoid accidental data loss, PostgreSQL starts complaining the needing for a `vacuum` when the next *xid* is coming near to 10 millions remaining values to the wrap around and deactivates itself when there is only 1 million left. If this threshold seems to much high to you please remember that each SQL statement is executed in a transaction context, even when a transaction has not been explicitly started.

Auto-Vacuum

Having to remember to manually vacuuming a cluster can be hard, and therefore, starting from the 8 series, PostgreSQL embeds an *auto-vacuum* feature. If auto-vacuum is enabled via the *autovacuum = on parameter* in the *postgresql.conf* file, and you wait enough time before executing the vacuum of Listing 7, you will see that such manual vacuum is doing almost nothing since the table has been already vacuumed. Autovacuum launches a set of *worker* processes every specified amount of time; each worker vacuums a table if it is long since last vacuum for the table or if the number of expired tuples is greater than a computed threshold. It is possible to set per-table autovacuum as in the following:

```
ALTER TABLE magazine SET (autovacuum_enabled = false);
```

Since vacuum could be a resource intensive operation, PostgreSQL provides a rich set of parameters for fine tuning of the auto-vacuum which are out the scope of this article.

Micro-Vacuum and HOT

Microvacuum is a page-boundary limited vacuum, which aim is to reclaim space within the same data page. Microvacuum is used in the HOT (Heap Only Tuple) subsystem: the idea is that if a tuple is modified only for out-of-index properties PostgreSQL should search to keep the new tuple version in the same data page, so to avoid an index update too. To do so, a microvacuum is done on the data page to free some space for the new version of the tuple, and then a pointers chain to the new tuple is placed to make the new version available to queries. Moreover, when a single data page is accessed (via `SELECT`, `UPDATE` or `DELETE`) a space cleanup is performed to keep the data page as much clean as possible.

Listing 7. *Cleaning up expired tuples using vacuum*

```
#!/bin/sh

DBOID='oid2name -q | grep bsdmagdb | awk '{print $1;}''
PAGE_SIZE='expr 8 '*' 1024'
TABLEOID='oid2name -U bsdmag -t magazine -d bsdmagdb -q | awk '{print $1;}''
FILENODE=${PGDATA}/base/${DBOID}/${TABLEOID}
echo "Cleaning the magazine table..."
ls -lh $FILENODE
SIZE_1='ls -lk $FILENODE | awk '{print $5;}''
psql -U bsdmag -c "VACUUM FULL VERBOSE magazine; " bsdmagdb
ls -lh $FILENODE
SIZE_2='ls -lk $FILENODE | awk '{print $5;}''
echo "=========================="
SIZE_1_T='expr $SIZE_1 / $SIZE_1'
SIZE_1_P='expr $SIZE_1 / $PAGE_SIZE'
SIZE_2_T='expr $SIZE_2 / $SIZE_1'
SIZE_2_P='expr $SIZE_2 / $PAGE_SIZE'
echo "Filenode was $FILENODE"
echo "Size before starting: $SIZE_1"
echo "Size after VACUUM:    $SIZE_2 [ $SIZE_2_P pages ] (around $SIZE_2_T times initial size)"
echo "=========================="
```

On The Web

- PostgreSQL official Web Site: *http://www.postgresql.org*
- ITPUG official Web Site: *http://www.itpug.org*
- PostgreSQL 9.1 Data Page Layout: *http://www.postgresql.org/docs/9.1/static/storage-page-layout.html*
- PostgreSQL 9.1 Documentation on Vacuum: *http://www.postgresql.org/docs/9.1/static/sql-vacuum.html*
- Bruce Momjan, MVCC Unmasked (Talk at the Fourth Italian PGDay): *momjian.us/main/writings/pgsql/mvcc.pdf*
- Scripts and examples used in this article are available via GitHub repository at *https://github.com/fluca1978/fluca-pg-utils*

In order to allow a better page-boundary vacuum, each table can have a specific *fillfactor*, that is percentage of free space to guarantee for updates. In particular the fillfactor specifies how much space can be consumed in a data page by `INSERT` commands, leaving the rest of the space free for `UPDATE`s. If a table is never updated the default fillfactor of 100 (full package) is the best, while if a table is often updated a lower fillfactor will preserve disk space in the long run. To see the effect of the fillfactor you can run again the script of Listing 6 setting a fillfactor of 40; the final result will be that the number of pages after the updates is the same as after the interts, since each page kept free space for new versions of the same tuples.

Summary and Coming Next

This article explained how PostgreSQL manages concurrency and how it stores tuples. Knowing how the internal storage works can be helpful in tuning and maintaning large database to perform at best.

In the next article readers will see how to replicate a running PostgreSQL cluster into another running instance.

LUCA FERRARI

Luca Ferrari lives in Italy with his wife and son. He is an Adjunct Professor at Nipissing University, Canada, a co-founder and the vice-president of the Italian PostgreSQL Users' Group (ITPUG). He simply loves the Open Source culture and refuses to log-in to non-Unix systems. He can be reached on line at http://fluca1978.blogspot.com.

Beowulf Clusters with DragonflyBSD

There are two types of computing clusters: High availability (HA) clusters are designed so that if one computer fails, the other(s) take over its job. HPC clusters enable many computers to do the same job together so that processing power is increased. We're going to focus on the latter.

What you will learn…

- How to build a high performance computing (HPC) cluster with DragonflyBSD

What you need

- The ability to use the command line interface.
- Two or more computers with DragonflyBSD 2.10.1 on the same subnet. Each computer must be running the same architecture (don't mix 32-bit with 64-bit).
- An understanding of what it means to compile a program.

An HPC cluster on consumer grade hardware is called a *Beowulf* after the classic poem written sometime between 700 – 1000 AD. Beowulf technology is the result of a 1994 cooperative research project between NASA and several universities. Since DragonflyBSD development focuses so much on performance, it seems the best option for a BSD Beowulf. In fact, HPC clusters are one of the stated design goals of DragonflyBSD.

There isn't a software program called Beowulf. There are several solutions for implementing Beowulf. We'll use a common solution called MPICH2. Fortunately, DragonflyBSD offers a package for MPICH2. In a Beowulf, one computer is the *master* node. It controls all of the other nodes called *clients*.

Let's start with our master node, which I've named wolfmaster. I know what you're thinking: *Beowulf has an 'u', and wolfmaster has an 'o'. You're inconsistent, Toby*. I know. I just felt like doing it that way. Wolfmaster has an IP address of 192.168.0.10. First, we install MPICH2:

```
# pkg_radd mpich2-1.3.1
```

Next, use the `adduser` command to add a user called wolf. Notice the UID number when you're done. Now that we have a `/home/wolf` directory we run a couple of commands. Users from the other node will use this directory, so we give wolf's profile a umask allowing access for other users.

```
# echo 'umask 007' >> /home/wolf/.profile
```

Export `/home/wolf` as an NFS share:

```
# echo '/home/wolf -alldirs -network 192.168.0.0 -mask
                    255.255.255.0' >> /etc/exports
```

To turn on NFS sharing at boot time, edit `/etc/rc.conf`, and add these lines:

```
portmap_enable="YES"
nfs_server_enable="YES"
mountd_flags="-r"
```

MPICH2 uses hostnames even if you tell it to use IP addresses, so if you don't have the names in a DNS server somewhere, you'll have to edit the hosts file like I did by adding the following lines:

```
192.168.0.10 wolfmaster wolfmaster.
192.168.0.11 wolfnode00 wolfnode00.
```

I gave each node an alias of its name as well as its name followed by a "." because MPICH2 wants to use the fully qualified domain name (FQDN). While installing DragonflyBSD, I did not provide an FQDN for the host name, so MPICH2 adds the "." when looking for other nodes.

In this article, we're building a cluster with only two nodes. Beowulf supports up to 1024 nodes. You wouldn't want to update 1024 hosts files each with 1024 entries. These next steps will become clear later on in the article:

```
# mv /etc/hosts /home/wolf/hosts
# ln -s /home/wolf/hosts /etc/hosts
```

That's all we need to do as root on wolfmaster. Now log in as the wolf user. We have a bit more work to do for password-free SSH.

Begin Adding Node

```
$ ssh-keygen -b 2048 -f ~/.ssh/id_rsa -t rsa -N „"
```

Note that after the `-N` argument, there are two double quotes with nothing between them, not even a space. This ensures that no passphrase will be required when doing remote login. Then copy the file `id_rsa.pub` into another file `authorized_keys`:

```
$ cd ~/.ssh
$ cp id_rsa.pub authorized_keys
$ chmod 644 ~/.ssh/authorized_keys
$ chmod 755 ~/.ssh
```

We could start NFS now by starting/restarting nfsd and mountd, but we want to be sure that NFS comes up at boot time. Let's restart wofmaster now. Time to move onto the client node I've named the client wolfnode00, and it has an IP address of 192.168.0.11.

Start by invoking the `adduser` command again. Name the user wolf. When prompted for the UID, type in the same number as the UID of the wolf user on the master node. Use `pkg_radd` to install MPICH2.

```
echo 'wolfmaster:/home/wolf /home/wolf nfs ro 0 0' >>
                    /etc/fstab
```

Remember when we moved the /etc/hosts file on wolfmaster to `/home/wolf/hosts`, and then created a symbolic link for `/etc/hosts`? In a moment, wolfnode00 is going mount wolfmaster's `/home/wolf` as it's own. Creating a link from an NFS mount won't work; the following magic will prevent us from having to edit the `/etc/hosts` file on any client nodes. The `/etc/hosts` file will be updated at the top of every hour:

```
# echo '0 * * * * root cp -f /home/wolf/hosts /etc/hosts'
                    >> /etc/crontab
```

We could mount the NFS export with the mount command, but again: we want to make sure that it does its thing at boot time. Restart wolfnode00. By running the `df` command, you should see that the last line reads:

```
wolfmaster:/home/wolf [some information about blocks]
                    /home/wolf
```

To add more nodes, first modify the `/home/wolf/hosts` file. Then start from the place in this article that says `BEGIN ADDING NODE`. Substitute *wolfnode01* ...02, 03, etc for *wolfnode00*. When you get to this point, you'll have successfully added another node.

End Adding Node

Back to the master node. Log on as wolf. Execute this command:

```
$ ssh wolfnode00 hostname
```

You should not have to enter a password, and you should find that the hostname (wolfnode00) of the client (not the master) is returned. The last thing to do before we can start testing MPICH2 is to create a file on `/home/wolf`. I called it *nodes*. The contents should be the name of each node with one node per line, like this:

```
wolfmaster
wolfnode00
```

Beowulf programs are executed with the mpiexec command. There are two switches that we need for basic usage:

- -f specifies the file name with the list of nodes. In my case, `/home/wolf/nodes`
- -n specifies the number of nodes to run a program on. If you specify a number that is greater than the number of nodes you have, then performance will decrease.

To test the setup, try this:

```
$mpiexec -f /home/wolf/nodes -n 2 hostname
```

The result should be:

Table 1. *MPICH2 compilers*

Compiler	Language
mpicc	C
mpicxx	C++
mpif77	Fortran

```
wolfmaster
wolfnode00
```

Notice that the `hostname` command ran independently on each node. That's why we have two results instead of one. For a program to run with the performance of our HPC cluster, it must be compiled with the MPICH2 libraries. MPICH2 includes utilities for doing this: Table 1.

Fortunately, there are example programs to try. Unfortunately, they don't come with DragonflyBSD's MPICH2 package. Let's download the source tarball for MPICH2:

```
$ curl -o mpich2-1.3.1.tar.gz \
http://www.mcs.anl.gov/research/projects/mpich2/downloads
/tarballs/1.3.1/mpich2-1.3.1.tar.gz
$ tar xvfz mpich2-1.3.1.tar.gz
$ cd mpich2-1.3.1
```

The default install of DragonflyBSD doesn't include Fortran support:

```
$ ./configure --disable-f77 --disable-fc
$ make
```

First, let's try one of the precompiled examples:

```
$ mpiexec -f /home/wolf/nodes -n 2 /home/wolf
/mpich2-1.3.1/examples/cpi
```

This should return pi. In order to really test that our cluster is faster than one computer, we'll need to compile one of the examples. The MPICH2 libraries are in mpich2-1.3.1/lib, so the command looks like this:

```
$ mpicc -o /home/wolf/icpi -L /home/wolf/mpich2-1.3.1
/lib /home/wolf/mpich2-1.3.1/examples/icpi.c
```

The icpi program will ask you to input how many times to run the pi algorithm. I am using two 2 GHz Core 2 Duo computers, and 10,000,000,000 (ten billion) turned out to be a good test. First I ran it without mpiexec.

```
$ /home/wolf/icpi
```

Result: 23.6 seconds. Let's now run it with mpicc, but on only one node:

```
$ mpiexec -f /home/wolf/nodes -n 1 /home/wolf/icpi
```

Result: 23.6 seconds. Now the moment you've been waiting for. Let's run it on both nodes:

```
$ mpiexec -f /home/wolf/nodes -n 2 /home/wolf/icpi
```

Result: 11.8 seconds... Success!

When building your Beowulf, the speed of your network is paramount. Consider my two-node Beowulf. Calculating pi doesn't use much memory. Everything happens in the caches of the CPU's. What about more memory intensive programs? My nodes each have an 800MHz *front side bus* (FSB). I'm running a 32 bit OS. Multiply that by my 32 registers, and you get 25.6 Gbps as opposed to the data transfer rate of my network: 1Gbps. More likely a Beowulf today would consist of computers with a 1.6GHz FSB and a 64 bit OS. Now our RAM is communicating with the CPU at 102.4 Gbps. I'm not even going to get into the effects of dual-, triple-, or quad-channel memory (mostly because the known benchmarks are inconclusive) *http://en.wikipedia.org/wiki/Multi-channel_memory_architecture#Performance*.

Clearly I'm better off using a computer with two CPU's instead of a two-node Beowulf where each node has a single CPU. Beowulf is useful because it's far less expensive to build a cluster with 20 computers instead of buying a single

Figure 1. *A 50 node Beowulf used for detecting and analyzing pulsars at McGill University*

For more information on Beowulf and MPICH2, check out the following web sites:

- Beowulf site (including the history of Beowulf): *http://www.beowulf.org*
- MPICH2 site: *http://www.mcs.anl.gov/research/projects/mpich2/*
- DragonflyBSD site: *http://www.dragonflybsd.org*
- Information about loss of a node: *http://mpich-v.lri.fr/papers/MPICH-V2.pdf*
- McGill Pulsar Group: *http://www.physics.mcgill.ca/~pulsar/Welcome.html*

computer with 20 CPU's; still, you can see why you wouldn't want to run a Beowulf on a slow network connection.

When is a Beowulf useful? When your program requires lots of processing power, but little disk access. A web server or database server would not be a good use of Beowolf. Those sorts of programs are more disk and memory intensive relative to the processing power that they need. Projects that require lots and lots of math computations are the best use for Beowulf. For this reason, you'll find Beowulfs in the science departments of many universities.

Let's wrap things up by talking about a real world example of why someone might want a Beowulf. A friend of mine named Brian has a degree in engineering, and he builds robots for fun. Typically, he'd build them for Battle Bots (*www.battlebots.com*). In 2008, he and his team decided to enter a competition sponsored by NASA to build a robot for moving dirt around on the Moon.

Robot building is an expensive hobby. Brian is always complaining about the price of titanium. Rather than build several prototypes, they had to get the design right the first time. They needed to run computer simulations to predict how the final product would perform. As amateurs, there's no way that he and his team could afford time on a supercomputer.

Fortunately for Brian, he has friends in the e-waste industry. Many organizations dispose of computers that still function. Brian was able to obtain about fifty working 2 GHz Core 2 Duo machines for free. With Beowulf, they were able to run the necessary simulations for building their robot at 2 GHz * 50 nodes = 100 GHz. Of eight competitors, Brian's team was the only robot that could complete the objectives of the competition.

TOBY RICHARDS

Toby Richards has been a network administrator since 1997. Each article comes straight from the notes that he takes when doing a new project with *BSD. Toby recommends bsdvm.com for your hosting needs because they provide console access to your virtual machine.

Npppd: Easy PPTP VPN with OpenBSD

Have you ever needed to set up a VPN for Microsoft Windows or Mac OS X users? From this article you will find out how to configure OpenBSD and npppd to provide PPTP and L2TP VPN's in a few easy steps.

What you will learn...

- How to setup a PPTP/L2TP VPN server

What you should know...

- Basic OpenBSD tasks
- Basic TCP/IP and routing knowledge

In January 2010, npppd was imported into the OpenBSD source tree and this software can act as a PPTP/L2TP VPN server and also as a PPPOE server.

Because npppd is still under active development and still missing some features, it is not linked to the standard build yet, so to install the program you first need to build it from OpenBSD source tree.

First install OpenBSD-current or wait for 5.1 which will be released around May 1st, 2012 (Listing 1).

Now you will have two new programs installed under `/usr/sbin: npppd`, the PPTP/L2TP daemon and npppctl, the userland utility.

First you want to configure the software: Enable the gre, esp and pipex protocols using sysctl: Listing 2.

Edit `/etc/sysctl.conf` accordingly to make these changes persistent across reboots.

Create a directory under `/etc` where you will put your configuration files:

```
$ sudo mkdir -m 0755 /etc/npppd
```

Create the `/etc/npppd/npppd.conf` configuration file: Listing 3.

With this example configuration, the tun0 interface is used to concentrate VPN access and the 192.168.255.0/25 network is assigned to users.

What's Wrong With Poptop?

Poptop has many features, runs on many platforms and can be installed with one simple command on OpenBSD: `pkg _ add poptop`.

However, Poptop is not designed with security in mind (it has no privilege separation), it does not provide RADIUS authentication (without unofficial patches to PPP),and, at least on OpenBSD, it does not perform very well.

Listing 1. *Install npppd*

```
$ cd /usr
$ cvs -d anoncvs@anoncvs.fr.openbsd.org:/cvs checkout -P
src/usr.sbin/{Makefile.inc,npppctl,npppd}
$ cd src/usr.sbin/npppd
$ make
$ sudo make install
$ cd ../npppctl
$ make
$ sudo make install
```

Listing 2. *Enable needed protocols*

```
$ sudo sysctl net.inet.gre.allow=1   #(for PPTP)
$ sudo sysctl net.inet.esp.allow=1   #(for IPSEC)
$ sudo sysctl net.pipex.enable=1     #(for PIPEX)
```

Listing 3. *Npppd configuration*

```
[npppd.conf]
--------------------------------------------------------------------------------
interface_list:              tun0
interface.tun0.ip4addr:      192.168.255.1

# IP address pool
pool.dyna_pool:           192.168.255.0/25
pool.pool:                192.168.255.0/24

# Local file authentication
auth.local.realm_list:          local
auth.local.realm.acctlist:      /etc/npppd/npppd-users.csv
realm.local.concentrate:        tun0

# RADIUS authentication / accounting
#auth.radius.realm_list:                        radius
#auth.radius.realm.server.address:     127.0.0.1:1812
#auth.radius.realm.server.secret:      password
#auth.radius.realm.acct_server.address:127.0.0.1:1813
#auth.radius.realm.acct_server.secret: password
#realm.radius.concentrate:      tun0

lcp.mru:           1400
auth.method:          mschapv2 chap
ipcp.dns_primary:        192.168.107.254
#ipcp.dns_secondary:             192.168.6.20
ipcp.nbns_primary:       192.168.107.254
#ipcp.nbns_secondary:            192.168.6.20
#ipcp.assign_fixed:              true
#ipcp.assign_userselect:         true

pptpd.enabled:         true
pptpd.ip4_allow:       0.0.0.0/0

# L2TP daemon
l2tpd.enabled:            true
l2tpd.ip4_allow:       0.0.0.0/0
l2tpd.require_ipsec:         false
```

Listing 4. *Setup vpn users*

```
Username,Password,Framed-IP-Address,Framed-IP-Netmask,Description,Calling-Id
user1,password1
user2,password2
```

It is better to choose a network setup for your VPN that is different from the one you are using for your internal interface, otherwise your VPN users could have problems reaching your LAN.

When npppd starts, it will setup a tun interface with the IP address 192.168.255.1 and will authenticate users by reading the password file `/etc/npppd/npppd-users.csv`. `npppd` could also authenticate users using a RADIUS server.

Once you have configured your LAN DNS (`ipcp.dns_primary`) and netbios over TCP/IP (`ipcp.nbns_primary`) servers, you can start adding users to the password file.

The password file `/etc/npppd/npppd-users.csv` is a CSV file with minimal information about users: Listing 4.

Only the user and password fields are required and the second and third fields are used to assign the same IP address every time a specific user connects to the VPN.

The password file should be kept secured with standard file access permissions, if you want more security radius should be used instead.

Listing 5. *pptp vpn client setup*

```
vpn_pptp:
  set device "!/usr/local/sbin/pptp --nolaunchpppd A.B.C.D"
  set authname user1
  set authkey password1
  set mppe 128 stateless
  disable protocomp
  deny protocomp
  disable ipv6cp
```

Listing 6. *Monitoring vpn usage*

```
# npppctl session brief
Ppp Id     Assigned IPv4   Username             Proto Tunnel From
---------- --------------- -------------------- ----- -------------------------
         6 192.168.255.62  giovanni             PPTP  host222-19-dyn.51-85-r.provider.it:49230

# npppctl session all
Ppp Id = 8
          Ppp Id                  : 8
          Username                : giovanni
          Realm Name              : local
          Concentrated Interface  : tun0
          Assigned IPv4 Address   : 192.168.255.62
          Tunnel Protocol         : PPTP
          Tunnel From             : host222-19-dyn.51-85-r.provider.it
          Start Time              : 2012/02/07 14:05:41
          Elapsed Time            : 6967 sec (1 hour and 56 minutes)
          Input Bytes             : 5233902 (5.0 MB)
          Input Packets           : 38364
          Input Errors            : 1 (0.0%)
          Output Bytes            : 27619691 (26.3 MB)
          Output Packets          : 46374
          Output Errors           : 26 (0.1%)
```

Listing 7. *Routing configuration in a nat setup*

```
inet 192.168.255.1
!route add 192.168.255.0/25 192.168.255.1
```

Listing 8. */etc/ipsec.conf configuration to let l2tp protocol work*

```
ike passive esp transport \
   proto udp from any to any port 1701 \
   main auth "hmac-sha" enc "3des" group modp2048 \
   quick auth "hmac-sha" enc "aes" \
   psk "password"
```

Your PPTP setup is now complete and you can start npppd with -d debug mode enabled or the -D daemon mode parameter. You can now try your VPN:

```
# /usr/sbin/npppd -D
```

You can connect to your VPN with Microsoft Windows, Mac OS X, *BSD or Linux.

For example, you can use pptp from ports to connect to your VPN.

To install pptp from ports:

```
$ sudo pkg_add pptp
```

edit `/etc/ppp/ppp.conf` (Listing 5).

Modify „A.B.C.D" to your VPN public IP address and user and password according to your setup.

Now you can connect to your VPN:

```
$ sudo ppp -ddial vpn_pptp
```

You can monitor connected users from your VPN server console using `npppctl`: Listing 6.

If necessary, you can even disconnect VPN users from the VPN server with the command npppctl clear all.

Tips & Tricks

If your VPN server is under NAT, you should setup a route to connect your LAN and your VPN networks.

```
# route add 192.168.255.0/25 192.168.255.1
```

To make these changes permanent, you should edit `/etc/hostname.tun0` and add to the file: Listing 7.

L2TP VPN Setup

Layer 2 Tunneling Protocol (L2TP) is a tunneling protocol used to support virtual private networks (VPNs). It does not provide any encryption or confidentiality by itself; it relies on an encryption protocol that it passes within the tunnel to provide privacy.

In OpenBSD /etc/ipsec.conf is used to setup encryption parameters (Listing 8).

IPsec is now configured and you can run the IPsec daemon:

```
# isakmpd -Kv
```

Execute ipsecctl to notify isakmpd of configuration changes.

```
# ipsecctl -f /etc/ipsec.conf
```

Caveats

npppd is still a work in progress and some features are still not implemented:
there is no proxyarp support so you cannot use the same network space for both LAN and VPN.
There is also no support for NAT-T IPsec so if your VPN server is under NAT, it will work only with the PPTP protocol.

GIOVANNI BECHIS

Giovanni Bechis lives in Italy with his wife and son.He is an OpenBSD developer and the owner of SnB, a software house which provides web design and hosting solutions based mainly on *BSD systems.

He can be found at http://www.snb.it.

For many years, Poptop has been the only software available to create PPTP VPN connections from Linux/BSD.

Poptop has some issues that lead a Japanese software house start writing their own PPTP/L2TP VPN server.

Anatomy

of a FreeBSD compromise (Part 4)

Continuing our security series, we will look at the vulnerabilities on our test network

What you will learn…
- How to use Nessus, exploitation tools and payloads

What you should know…
- BSD and network administration skills

From the last article, we discovered that to penetrate a system we continually needed to move from the general to the specific, and to identify the most vulnerable system on our network depending on what services were running on it (Figure 1). We are now going to attempt to run a successful exploit on the machine with the most ports open, and to improve our chances, we will use a legacy version of FreeBSD 6.1 with Apache 1.3.37 rather than the current release.

Aim of the attack

From the hackers perspective, the aim is to release either a Zero Day Vulnerability (2) or a well documented proven attack (4) against an unprotected target host. A well protected or patched machine will not be vulnerable either because there is no known attack (1) or the machine has been patched against exploits previously found in the wild (3) – see (Figure 2). An undiscovered attack is low risk as the hackers are not aware of it, and at the same time as code is reviewed and tested, developers will (hopefully) preempt obvious holes. The highest risk is where a known attack is in the wild, yet the system is not patched or modified to counter this (4). The security footprint of the system is crucial – it can be argued that there are many

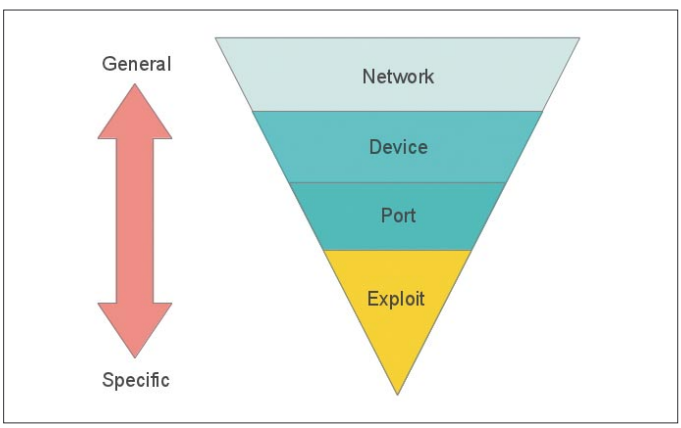


Figure 1. *Attack strategy*

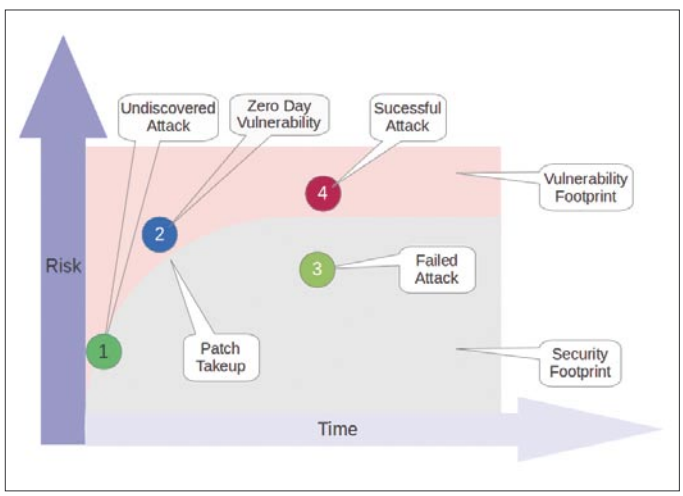


Figure 2. *Vulnerability window*

Table 1. *Potential Targets*

Potential Targets			
No	Hostname	IP Address	Discovered by
1	Hacker	192.168.0.131	NTOP, TCPDUMP, NMAP
2	Intel	192.168.0.132	NTOP, TCPDUMP, NMAP
3	?	192.168.0.141	TCPDUMP
4	?	192.168.0.250	NMAP
5	Border	192.168.0.254	NTOP, TCPDUMP, NMAP

Table 2. *Open Ports on victim*

Open ports on 192.168.0.254
25/tcp open smtp 53/tcp open domain 80/tcp open http 110/tcp open pop3 139/tcp open netbios-ssn 143/tcp open imap 587/tcp open submission 631/tcp open ipp 3128/tcp open squid-http

more exploits for certain platforms because they are more popular, so it will be interesting to see if the hackers develop more Linux / *BSD attacks with the growth in popularity of mobile devices.

Methodology

The potential types of attack include *Cross-Site Scripting* (XSS), Brute Force Password, Buffer Overflows, SQL Injection, Denial of Service etc. There are many ways of delivering a payload via shellcode, including C / Bash / Perl, custom scripts, executables etc. as well as using tools such as Metasploit. There is no *magic bullet* – a single tool or program that will guarantee a successful attack under all circumstances as there are too many variables to consider. A script-kiddie may pick up a file that gains access to some machines, but this may be ineffective across versions, patches, architectures or potentially even language versions or the amount of memory installed on the machine. In other words, *off the shelf* attacks are just the starting point for the serious hacker, as each exploit has to be carefully tuned to the victims' environment (Figure 3). The vulnerability belongs to the system, the exploit and payload to the attacker. While the exploit *opens the door* to the attacker, the payload does the actual *damage*. As the number of available exploits and payloads increase, it becomes very time consuming to quickly find the *best* vulnerability. This

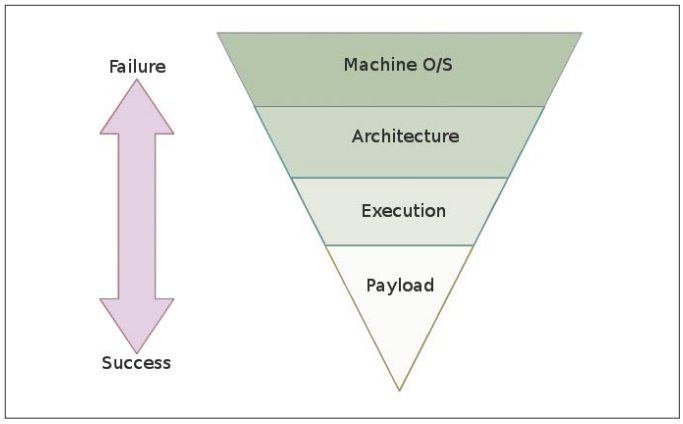


Figure 3. *Key to a successful exploit*

Table 3. *Further reading and resources*

Further reading	
Description	URL
Metasploit tutorial (Functionality may differ slightly from BSD)	*http://www.offensive-security.com/metasploit-unleashed/Main_Page*
Enabling Nessus on Backtrack 5	*http://blog.tenablesecurity.com/2011/07/enabling-nessus-on-backtrack-5-the-official-guide.html*
Obtaining a Nessus activation code	*http://www.tenable.com/products/nessus/nessus-plugins/obtain-an-activation-code*
Nmap website, lots of resources	*http://seclists.org*
Dictionary of publicly known information security vulnerabilities	*http://cve.mitre.org*
U.S. government repository of standards based vulnerability management data	*http://web.nvd.nist.gov/view/vuln/search*
Global cooperative cyber threat / internet security monitor and alert system	*http://isc.sans.org*
Backtrack 5 website	*http://www.backtrack-linux.org*
World class information security training and penetration testing	*http://www.offensive-security.com*
Exploit vulnerability database	*http://www.exploit-db.com*
Carnegie Mellon University's Software Engineering Institute	*http://www.cert.org*

Table 4. *mfsconsole useful commands*

mfsconsole examples	
Description	Command
Search for exploits or modules e.g. apache	Search apache
Show information for exploit	info
Show all exploits	show exploits
Show all payloads	show payloads
Show options for current exploit or module	show options
Load payload linux/x86/exec	set PAYLOAD linux/x86/exec
Show help	help

is where Nessus and Metasploit make such formidable tools in the security professionals' armory, as they open the door to automated vulnerability discovery and attack. The latest version of Backtrack (5R1) includes Armitage, a Java GUI driven attack management tool which greatly aids in discovering and executing vulnerabilities depending on the O/S type and the network operating environment. It will attempt to find the best match of exploit to victim. Metasploit also provides the `db_autopwn` utility which offers similar functionality.

Virtual Images or FreeBSD install?

With the availability of reliable desktop virtualization, *Ready Baked* versions of security toolkits such as Backtrack are available as Vmware images and ISOs. One other benefit of releases such as Backtrack is the wide inclusion of other tools not available on the *BSD platform such as Maltego, which allows the forensic investigator to build an accurate picture of the environment and data mine the results. At the time of writing, Backtrack R5 also supports the latest release of Nessus, whereas Backtrack R5r1 does not. Tenable Security document on their blog how to run Nessus under Backtrack, but this functionality is missing from Backtrack R5r1 for some reason (Table 3).

Figure 4. *Nessus under Backtrack*

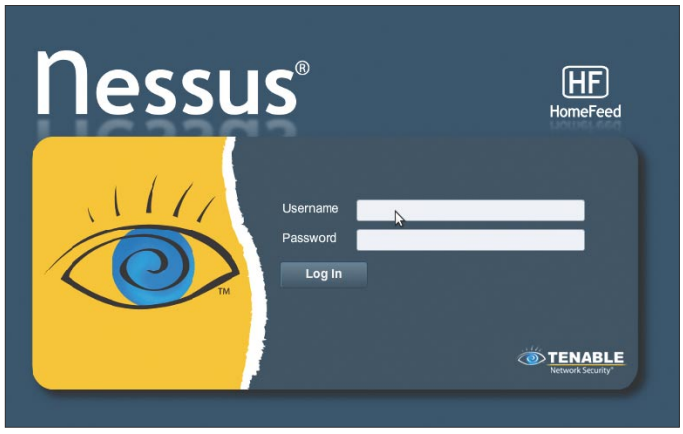


Figure 5. *Logging in to Nessus*

Installing tools such as Metasploit, Nessus etc. under *BSD is not quick – while packages are available, it is best to use ports. Although this is not difficult for experienced *BSD users, it is time-consuming to compile all the dependencies from scratch. For this tutorial I will be using a combination of Backtrack 5 (+Nessus), Backtrack 5r1 (+Armitage), and FreeBSD 9.0 running under Virtualbox.

To install Metasploit under FreeBSD 9.0 (as root):

```
pkg_add -r ruby
pkg_add -r ruby18-iconv
cd /usr/ports/security/metasploit
make install clean
cd /usr/local/share/metasploit
svn upgrade
msfconsole
svn up
```

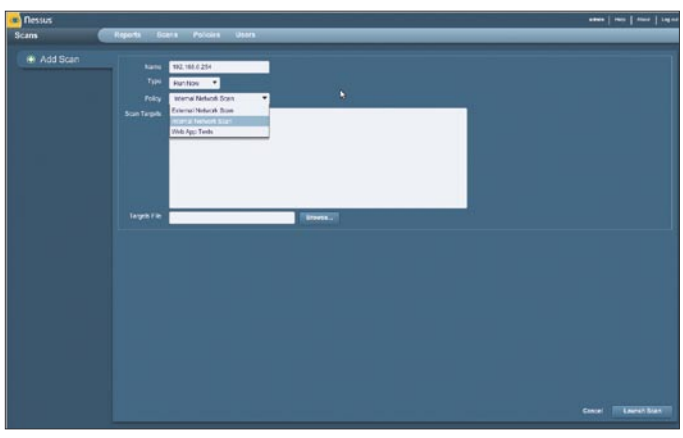


Figure 6. *Adding a scan for 192.168.0.254*

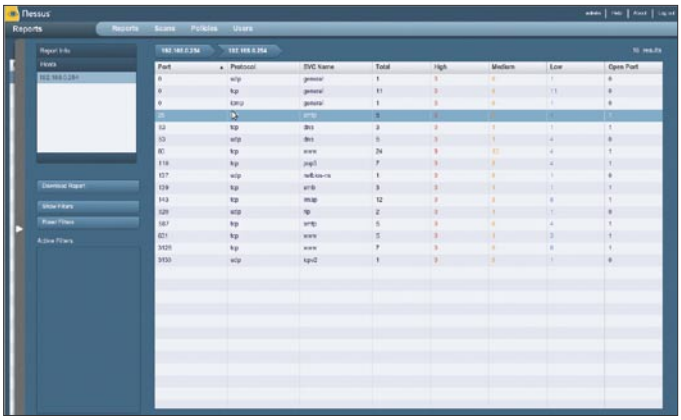


Figure 7. *Overview of vulnerabilities for 192.168.0.254*

This took > 50 minutes on my VM. Type exit to leave Metasploit. If you require DB support, this will need to be added separately either using `pkg _ add` or compiling from source.

Step 1 – Identify exploits

The quality of up to date information is critical when planning an attack. There are many useful security sites on the World Wide Web, but one of the the best tools to automate vulnerability discovery and assessment is Nessus. While a feed (which contains all the latest vulnerabilities) for home use is provided for free, if you wish to use Nessus in a commercial environment you must purchase a professional feed to meet the terms of their license. The other alternative is to manually research vulnerabilities, or to use the tools available with Metasploit and Armitage. There are other tools available with widely-ranging abilities, so Your Mileage May Vary (Table 3).

If you want to use Nessus, either boot from the Backtrack ISO or virtual image. You will be asked for a login name and password which is root and toor respectively. Follow the instructions to startx and you will be presented with the standard Gnome interface.

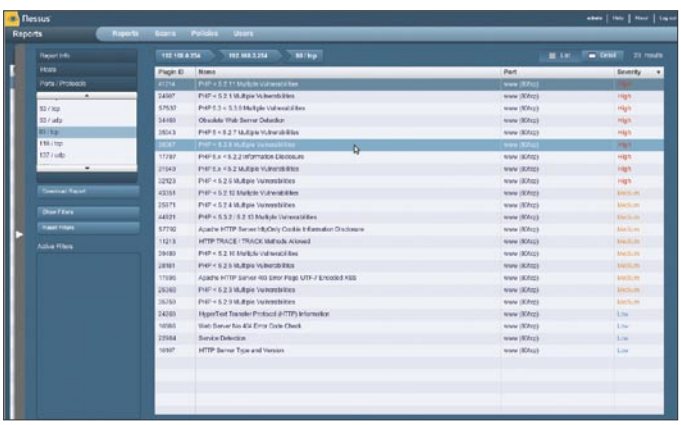


Figure 8. *HTTP vulnerabilities*

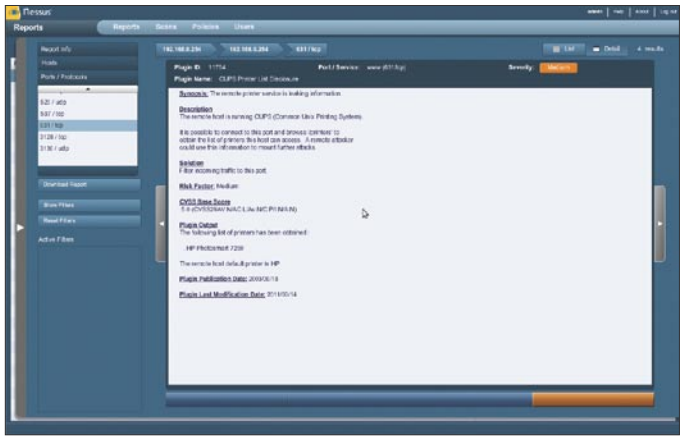


Figure 9. *Even the printer is not immune*

Click on Nessus Register to register for a home feed via your browser (Figure 4). An activation code will be sent to you via email. When you receive this, open a terminal and type the following to register:

```
/opt/nessus/bin/nessus-fetch -register @xxxx-xxxx-xxxx-
                    xxxx-xxxx@
```

Where `xxxx-xxxx-xxxx-xxxx-xxxx` is your authorization key. Now add a user with admin rights:

```
/opt/nessus/sbin/nessus-adduser
```

You may update your nessus plugins with the following command:

```
/opt/nessus/sbin/nessus-update-plugins
```

Start Nessus from the menu using Nessus Start (Figure 4). You can now login to Nessus either using the Firefox browser supplied in Backtrack, or via another machine. Point your browser at *https://localhost:8834* and login with the credentials you supplied earlier (Figure 5). There may be a delay while Nessus loads all the plugins. Add the target host you want to scan under internal network scan and Launch scan (Figure 6). After a while you will be presented with a set of reports that you can drill down through. From our test, we can see that the web-server

Figure 10. *Running Armitage*

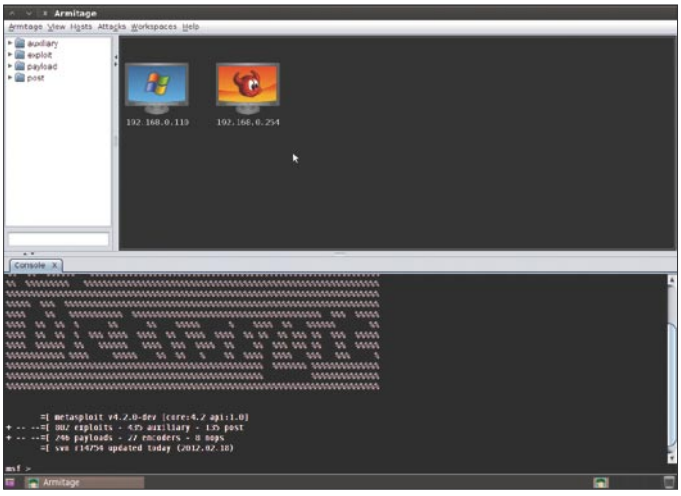


Figure 11. *Armitage GUI with hosts added and scanned*

is the most vulnerable with 23 vulnerabilities, 9 serious (Figure 7 – 9).

Step 2 – Metasploit

Metasploit runs pretty much identically under FreeBSD as it does under Backtrack, so we will use the BSD version to perform a test exploit for open telnet ports. Ensure your exploits etc. are up to date, and execute the following, replacing 192.168.0.0 with your network:

```
msfconsole

use auxiliary/scanner/telnet/telnet_encrypt_overflow
set RHOSTS 192.168.0.0/24
set THREADS 50
run
```

If you have a vulnerable device on your network, it will be shown as vulnerable.

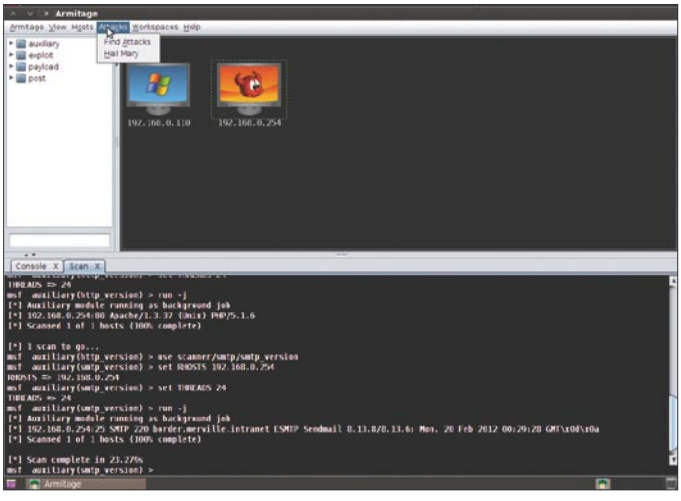


Figure 12. *Finding attacks*

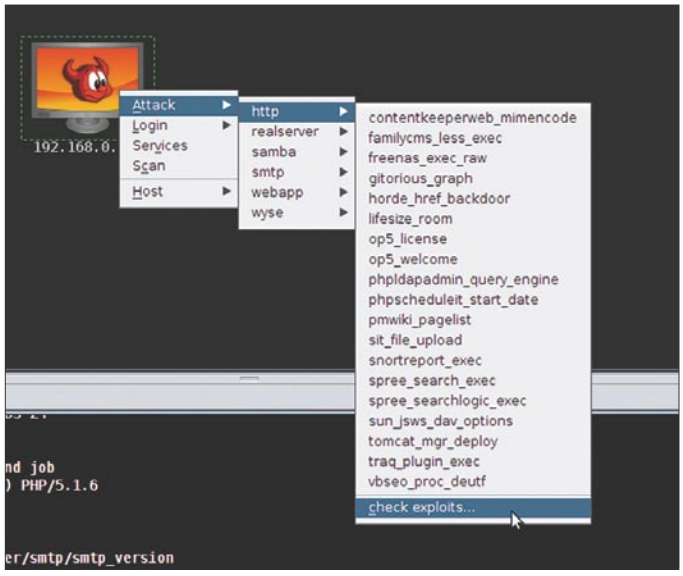


Figure 13. *checking for http exploits*

Tips

Useful msfconsole commands are listed in Table 4. A good exploit to try on *BSD is `exploit/freebsd/telnet/telnet_encrypt_keyid` as it is new and has an excellent chance of success. As the framework is script based, each script will require different parameters, info and help are your friends.

Step 3 (Optional) – Running Armitage

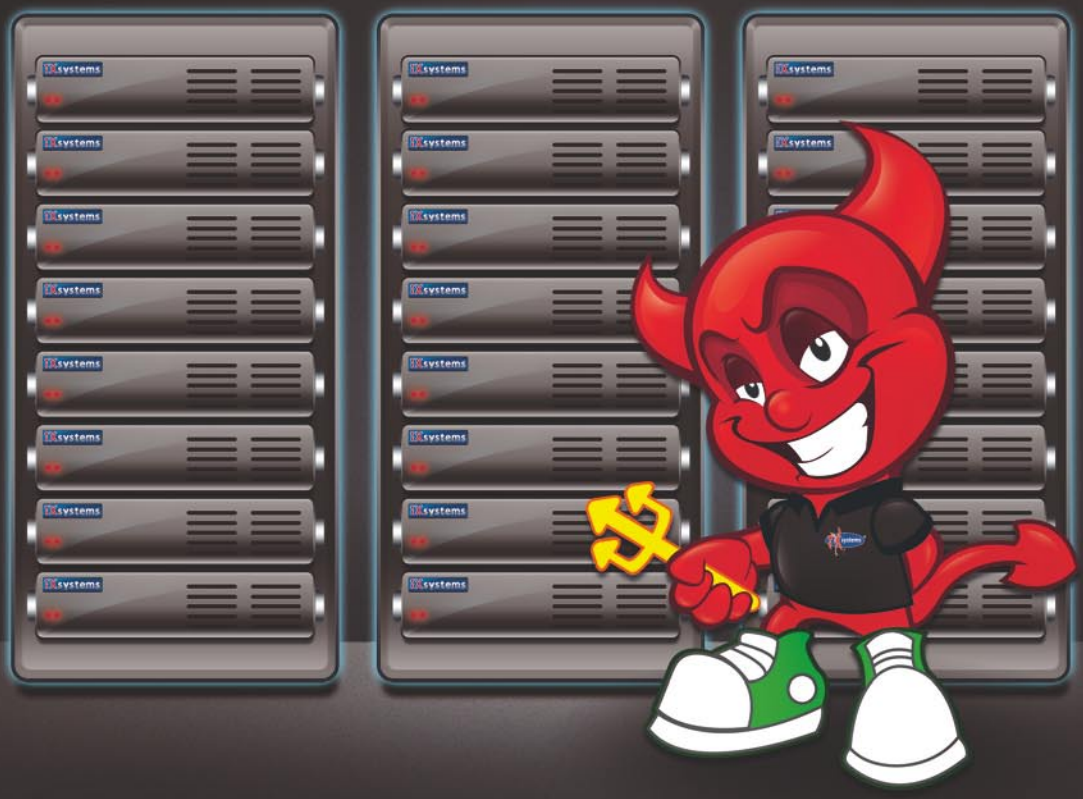
Using the Backtrack 5r1 ISO, login as above and run Armitage (Figure 10). Login to the server with the provided username and password, and run RPC support when requested. After a short delay, you will be presented with the Armitage GUI (Figure 11). Add the hosts you want to test via the hosts menu, and right click the PC to scan and update the O/S type if known. Find attacks for these devices on the Attack menu, and when right clicked you can run all the exploits (e.g. http) in one go by running *check exploits...* against the host (Figure 12-13). If an attack is successful, it will change color and this will be shown in the console at the bottom. To hunt for exploits across all devices, use the Hail Mary option.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.


Looking for help, tip or advice?
Want to share your knowledge with others?
BSD MAGAZINE
Give us your opinion about the magazine's content
and help us create the most useful source for you!

What has your server vendor done for BSD lately? Probably, not much.



Work with a vendor that supports the operating system you love!

iX is the corporate sponsor of the PC-BSD® Project, a major corporate donor to the FreeBSD Foundation, and leads the FreeNAS™ development team -- all while employing some of the most brilliant minds in the FreeBSD® community. For BSD hardware and software expertise, look no further.

1-855-GREP-4-IX

http://www.iXsystems.com/community

iX systems®